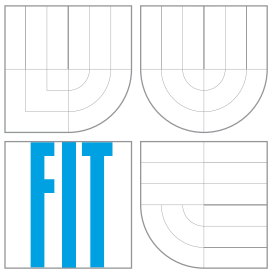
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZHRANÍ SYSTÉMU PRO ANALÝZU DOKUMENTŮ S JAVASCRIPTOVÝM KLIENTEM

INTERFACE FOR A DOCUMENT ANALYSIS SYSTEM WITH A JAVASCRIPT CLIENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ANDREA MARCELYOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2016

Zadání diplomové práce/18105/2015/xmarce02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Marcelyová Andrea, Bc.**
Obor: Informační systémy
Téma: **Rozhraní systému pro analýzu dokumentů s JavaScriptovým klientem**
Interface for a Document Analysis System with a JavaScript Client
Kategorie: Web

Pokyny:

1. Prostudujte existující rámce pro tvorbu klientských aplikací ve webovém prohlížeči v jazyce JavaScript, jako je např. Angular.js.
2. Seznamte se s platformou Java se zaměřením na tvorbu serverové části webové aplikace.
3. Seznamte se s rámcem FITLayout pro segmentaci a analýzu dokumentů a s jeho aplikačním rozhraním.
4. Po dohodě s vedoucím navrhněte architekturu nástroje s webovým rozhraním poskytujícím obdobné funkce, jako stávající grafické nástroje dostupné ve FITLayout.
5. Implementujte navržený nástroj.
6. Proveďte testování vytvořeného řešení a porovnejte funkčnost s existujícími v balíku FITLayout.
7. Zhodnoťte dosažené výsledky.

Literatura:

- Ondřej Žára: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015
- Josh Juneau: Java EE 7 Recipes, Apress, 2013
- Dokumentace k projektu FITLayout: http://www.fit.vutbr.cz/~burgetr/FITLayout/

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 4

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Táto diplomová práca sa zaoberá preštudovaním rámcov pre tvorbu klientskych aplikácií, platformy Java, desktopového nástroja FITLayout, návrhu architektúry nástroja s webovým rozhraním, popisom implementácie webovej aplikácie, testovaním a porovnaním funkčnosti desktopového a webového nástroja. Sú tu popísané existujúce rámce pre vývoj klientskych aplikácií vo webovom prehliadači v jazyku JavaScript, platforma Java so zameraním na tvorbu serverovej časti webovej aplikácie, rámec FITLayout pre segmentáciu a analýzu dokumentov. Nasleduje návrh architektúry nástroja s webovým rozhraním poskytujúcim podobné funkcie, ako súčasné grafické nástroje dostupné v nástroji FITLayout. Následne sú popísané významné aspekty implementácie. Z závere je popísaný priebeh testovania a porovnanie funkčnosti desktopového nástroja a implementovanej webovej aplikácie.

Abstract

This thesis deals with the study of existing frameworks for creating client-side applications, the Java platform, FITLayout software and software architecture design with a web interface, implementation of a web application, testing and comparing the functionality of desktop software with web application. Existing frameworks for the development of client-side applications in JavaScript programming language are described as well as the Java platform focused on creating a web application server component, FITLayout software for segmentation and analysis of documents. Software architecture design providing similar functionality such as existing graphical tools available in FITLayout and important aspects of implementation follows. At the end of the thesis testing of web application is described as well as comparison of desktop FITLayout software and web application that was implemented.

Kľúčové slová

JavaScriptové rámce, FITLayout, Angular, Java, REST

Keywords

JavaScript frameworks, FITLayout, Angular, Java, REST

Citácia

MARCELYOVÁ, Andrea. *Rozhraní systému pro analýzu dokumentů s JavaScriptovým klientem.* Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

Rozhraní systému pro analýzu dokumentů s JavaScriptovým klientem

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracovala samostatne pod vedením pána Ing. Radka Burgeta, Ph. D. Uviedla som všetky literárne pramene a publikácie, z ktorých som čerpala.

.....
Andrea Marcelyová
24. mája 2016

Poďakovanie

Rada by som poďakovala vedúcemu diplomovej práce pánovi Ing. Radkovi Brugetovi za ústretový prístup, hodnotné rady a odborné vedenie počas mojej práce.

Obsah

Kapitola 1

Úvod

Porozumenie obsahu webových stránok predstavuje veľkú výzvu počítačovej vedy. Aby bolo možné porozumieť obsahu webových stránok, je nutné nájsť spoľahlivý spôsob ako ich analyzovať. Existuje veľké množstvo ciest, ako k tejto problematike pristúpiť, a preto je vhodné mať k dispozícii nástroj, ktorý umožní jednoducho rôzne prístupy overovať na reálnych dokumentoch.

Pre potreby výskumu analýzy dokumentov bol na Fakulte informačních technológií Vysokého Učení Technického vytvorený nástroj FITLayout. V súčasnej dobe existuje desktopová verzia nástroja s grafickým užívateľským rozhraním. Desktopové aplikácie však ustupujú do úzadia a naopak stále väčšiu popularitu dosahujú webové aplikácie. Medzi výhody webových aplikácií patrí rýchla dostupnosť vo webovom prehliadači bez nutnosti inštalácie špeciálneho softwaru. Bolo by preto vhodné vytvoriť webovú aplikáciu, ktorá bude ponúkať podobné funkcie ako súčasná desktopová verzia nástroja FITLayout.

Cieľom tejto diplomovej práce je preskúmať rôzne technológie určené pre vývoj klientskych aplikácií vo webovom prehliadači v programovacom jazyku JavaScript, podobne preskúmať možnosti tvorby serverovej časti webových aplikácií v programovacom jazyku Java. V ďalšom kroku nasleduje zoznámenie sa s nástrojom pre analýzu a segmentáciu dokumentov FITLayout, ďalším cieľom je návrh architektúry nástroja s webovým rozhraním, ktorý bude poskytovať obdobné funkcie ako súčasná desktopová verzia nástroja FITLayout. Nasleduje implementácia navrhnutého nástroja a na záver testovanie vytvoreného riešenia a porovnanie s existujúcou desktopovou verziou nástroja.

V úvode nasledujúcej kapitoly 2 je možné nájsť stručnú charakteristiku programovacieho jazyka JavaScript a stručný popis architektonického vzoru Model-View-Controller a jeho modifikácií. V podkapitolách sa nachádza popis niektorých vybraných rámcov slúžiacich k tvorbe klientskych aplikácií vo webovom prehliadači v programovacom jazyku JavaScript. K jednotlivým rámcom je uvedená charakteristika s vybranými významnými vlastnosťami. Súčasťou popisu jedného rámca je aj jednoduchá ukážka použitia daného rámca.

V kapitole 3 sa nachádza stručná charakteristika programovacieho jazyka Java, ďalej popis architektonického štýlu Representational State Transfer. V podkapitolách je možné zoznámiť sa s dvoma vybranými rámcami pre tvorbu serverovej časti webových aplikácií. Súčasťou ich charakteristiky sú aj stručné ukážky použitia.

Ďalšia kapitola 4 predstavuje nástroj pre analýzu a segmentáciu stránok FITLayout. Je tu popísaná jeho základná architektúra, moduly a druhy služieb, ktoré nástroj umožňuje vytvárať. Nasleduje popis princípu vykresľovania dokumentov a charakteristika procesu segmentácie. Súčasťou kapitoly je aj zoznámenie sa s grafickým užívateľským rozhraním nástroja. Na konci kapitoly je možné nájsť zhodnotenie súčasného stavu a plán práce.

Návrh aplikácie sa nachádza v ďalšej kapitole 5. V prvej podkapitole je možné vidieť grafický návrh užívateľského rozhrania klientskej webovej aplikácie. Následne je uvedená architektúra nástroja spolu s náčrtom tejto architektúry vrátane jednotlivých vrstiev. Nakoniec je priblížený návrh klientskej a serverovej časti spolu s výpisom a stručnou charakteristikou služieb, ktoré bude server poskytovať.

V kapitole 6 je možné vidieť popis implementácie výslednej aplikácie. V prvej podkapitole sa nachádza popis technológií, ktoré boli pri implementácii aplikácie použité. Nasleduje predstavenie vytvorenej klientskej aplikácie FITLayout4Web spolu s jej ukážkami a vymenovaním jej funkcií. V ďalšej podkapitole je priblížená problematika implementácie klientskej časti, nachádza sa tu popis jednotlivých komponent tvoriacich grafické užívateľské rozhranie a schematická ukážka vykresľovania dokumentov. V závere sú podobne uvedené podstatné prvky implementácie serverovej časti aplikácie.

V predposlednej kapitole 7 je popísaný proces testovania výslednej klientskej aplikácie, popis rozdielov medzi výsledkami implementovanej klientskej aplikácie a existujúcej desktopovej verzii nástroja FITLayout. V závere kapitoly sa nachádza ukážka grafického užívateľského rozhrania obidvoch nástrojov a porovnanie funkčnosti týchto nástrojov.

V záverečnej kapitole 8 sa nachádza zhrnutie dosiahnutých výsledkov a náčrt ďalších možností pokračovania práce.

Kapitola 2

Rámce pre tvorbu webových aplikácií

V tejto kapitole je predstavený programovací jazyk JavaScript, stručne popísané architektonické vzory Model-View-Controller a jeho modifikácie, vybrané frameworky (rámce, ďalej bude využívaná len anglická forma, pretože je to zvykom aj medzi odborníkmi v obore) pre tvorbu klientskych aplikácií vo webovom prehliadači v jazyku JavaScript. Konkrétne sú predstavené frameworky AngularJS v podkapitole 2.1, Backbone.js v podkapitole 2.2, Ember.js v podkapitole 2.3, React v podkapitole 2.4 a nakoniec framework Knockout.js v podkapitole 2.5. Tieto frameworky uľahčujú prácu a pomáhajú udať štruktúru kódu na klientskej strane. Nejedná sa o úplný prehľad, sú popísané iba niektoré aspekty vybraných frameworkov.

Skriptovací programovací jazyk **JavaScript** [5] je používaný najmä pri tvorbe webových stránok. Bol vytvorený v roku 1995 a jeho autorom je Brendan Eich. Pôvodným názvom bol jazyk Mocha, neskôr LiveScript a nakoniec využívajúc vtedajšej popularity programovacieho jazyka Java bol stanovený názov JavaScript.

JavaScript spolu so značkovacím jazykom HTML[1] a kaskádovými štýlmi CSS[2] tvoria trojicu základných technológií tvorby obsahu webových stránok. Väčšina dnešných webových stránok používa JavaScript a je podporovaný všetkými modernými prehliadačmi. Je tiež používaný aj v prostrediach, ktoré nie sú webové, ako napríklad PDF dokumenty alebo desktopové grafické komponenty. Novšie a rýchlejšie virtuálne stroje zvýšili popularitu JavaScriptu pre serverovú časť webovej aplikácie. Na klientskej strane bol JavaScript tradične implementovaný ako interpretovaný jazyk, ale väčšina webových prehliadačov vykonáva kompiláciu za behu (angl. just-in-time). Je tiež používaný pri vývoji hier, tvorbe desktopových a mobilných aplikácií alebo pre serverovo orientované sieťové programovanie s behovými prostrediami ako napr. Node.js[3].

Frameworky pre tvorbu klientskych aplikácií vo webovom prehliadači v jazyku JavaScript sú často používané pre vytváranie jednostránkových webových aplikácií (angl. single-page (SPA)). SPA je webová aplikácia, ktorá je načítaná do webového prehliadača a reaguje na zmeny na klientskej strane bez nutnosti kompletného obnovenia webovej stránky zo servera.

Všetky nižšie uvedené frameworky používajú niektorý z architektonických vzorov Model-View-Controller alebo jeho modifikácie Model-View-Presenter a Model-View-ViewModel. Tieto architektonické vzory sú používané pre implementáciu užívateľských rozhraní. Architektonický vzor rozdeľuje danú softwarovú architektúru na tri prepojené časti - dátový model aplikácie, užívateľské rozhranie a riadiacu logiku tak, aby modifikácia v niektorej časti mala minimálny dopad na ostatné dve časti. Pôvodne bol návrhový vzor Model-View-Controller

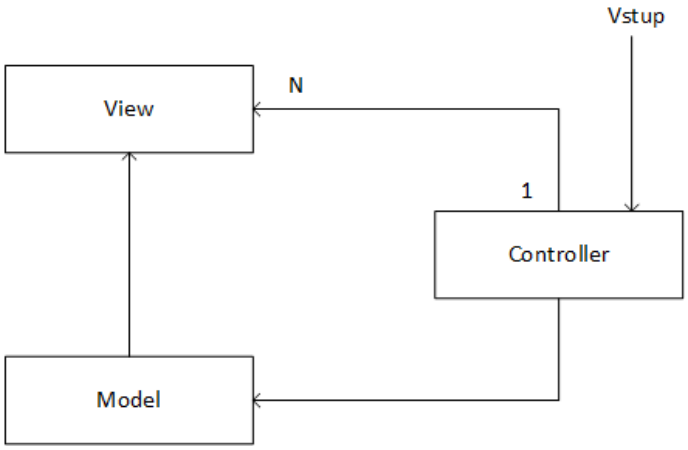
[1] https://html.spec.whatwg.org/
[2] http://www.w3.org/Style/CSS/
[3] https://nodejs.org/en/

vytvorený pre tvorbu desktopových aplikácií, ale rozšíril sa aj ako architektúra internetových aplikácií vo väčšine programovacích jazykov. Podobne ako v prípade iných softwarových vzorov, aj tento architektonický vzor predstavuje jadro riešenia problému s možnosťou prispôsobenia sa každému systému. Konkrétne architektúry sa môžu vzájomne mierne líšiť.

Stručná charakteristika architektonických vzorov Model-View-Controller a jeho modifikácií Model-View-Presenter a Model-View-ViewModel:

- **Model-View-Controller** (MVC), viď obrázok 2.1, je softwarová architektúra, ktorá rozdeľuje dátový model aplikácie, užívateľské rozhranie a riadiacu logiku do troch nezávislých komponentov a to tak, aby modifikácia v jednom komponente mala iba minimálny vplyv na ostatné dva komponenty [12].

  Model je komponent, ktorý sa stará o logiku aplikačných dát, View je komponent, ktorý prevádza dáta reprezentované modelom do podoby vhodnej na prezentáciu užívateľovi a komponent Controller reaguje na udalosti a zaisťuje zmeny v modeli alebo pohľade.



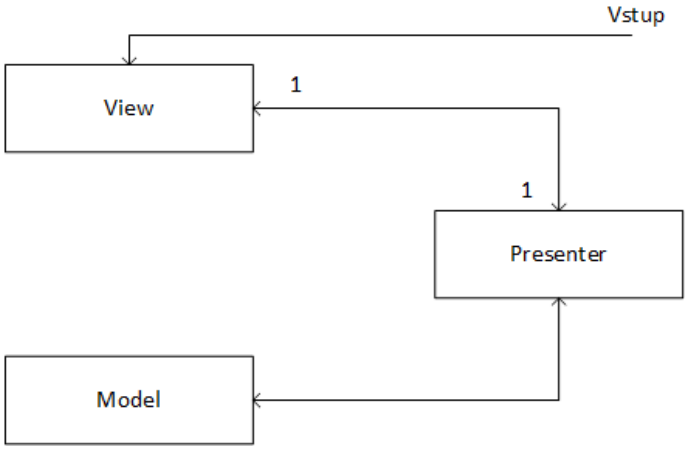
Obr. 2.1: Model-View-Controller

  Ak užívateľ vykoná nejakú udalosť v užívateľskom rozhraní, komponent Controller obdrží oznámenie o tejto udalosti. Udalosťou môže byť napríklad stlačenie tlačítka. Medzi komponentom View a komponentom Controller je vzťah 1:N, pretože Controller môže na základe vykonanej operácie vybrať rôzne pohľady (Views). Naopak komponent View nevie o komponente Controller a ani na neho nemá referenciu. Komponent Controller potom pristúpi k modelu a v prípade potreby ho aktualizuje. Komponent View použije aktualizovaný model pre zobrazenie aktualizovaných dát užívateľovi.

- **Model-View-Presenter** (MVP), viď obrázok 2.2, je modifikácia architektonického vzoru MVC [12].

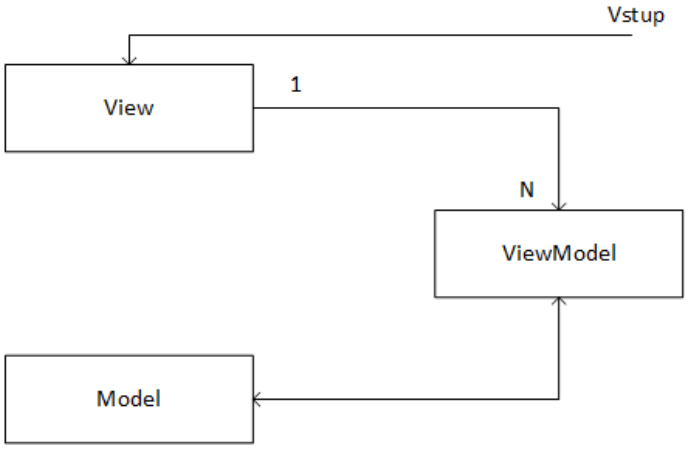
  V tomto prípade vstup začína v komponente View. Vzťah medzi komponentmi View a Presenter je 1:1. Komponent View má referenciu na komponent Presenter a tiež komponent Presenter reaguje na udalosť spustenú komponentom View. Na základe zmien v komponente Model sú dáta aktualizované komponentom Presenter.

  Hlavný rozdiel oproti architektonickému vzoru MVC je, že v prípade MVC má komponent View väčšiu kontrolu nad užívateľským rozhraním a obsluhuje udalosti, zatiaľ čo komponent Presenter vo vzore MVP je viac pasívny a iba prezentuje informácie cez užívateľské rozhranie.



Obr. 2.2: Model-View-Presenter

- **Model-View-ViewModel** (MVVM), viď obrázok 2.3, je opäť derivát architektonického vzoru MVC [10].



Obr. 2.3: Model-View-ViewModel

V tomto vzore je vzťah komponentu View a komponentu ViewModel 1:N. Rôzne pohľady môžu zdieľať ten istý ViewModel. Komponent View tiež nevie o existencii komponentu Model a tým pádom za svoj Model považuje komponent ViewModel.

2.1 AngularJS

Framework AngularJS[4] bol vytvorený v roku 2009, jeho zakladateľom je Miško Hevery a v súčasnosti je vyvíjaný spoločnosťou Google [6]. Angular bol navrhnutý tak, aby pomáhal oddeľovať zobrazovaciu logiku od aplikačnej logiky. Pre tento účel využíva framework architektonické vzory MVC aj MVVM. Vďaka tomuto oddeleniu je väčšina logiky obsiahnutá v časti Model alebo časti Controller.

Značkovací jazyk HTML je veľmi užitočný pri implementovaní statickej časti aplikácií, ale jeho možnosti implementácie dynamických prvkov webovej aplikácie sú obmedzené. Angular

[4] https://angularjs.org/

preto umožňuje vlastné rozšírenie tzv. HTML slovníku. Výsledné prostredie je potom vďaka tejto možnosti ľahko čitateľné a rýchlo rozšíriteľné.

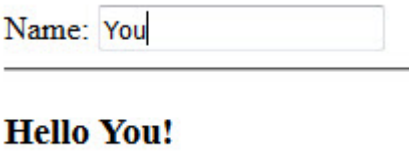
Angular umožňuje obojsmernú synchronizáciu dát (angl. data-binding). To znamená, že komponent View sa obnoví kedykoľvek nastane zmena komponentu Model a naopak obnovenie komponentu Model nastane vždy pri zmene komponentu View. Veľkou výhodou tejto funkcie z pohľadu programátora je eliminácia manipulácie s objektovým modelom dokumentu (angl. Document Object Model (DOM)).

Ďalšou vlastnosťou Angularu je podpora vkladania závislostí (angl. Dependency Injection (DI)). Závislosť je objekt, ktorý je možné využiť. DI je návrhový vzor, ktorého podstatou je vkladanie závislostí objektom externou autoritou namiesto toho, aby si objekty obstarali závislosti sami.

Na nasledujúcej ukážke HTML kódu je možné vidieť veľmi jednoduchú ukážku použitia frameworku Angular verzia 1.5.5.

```
<!doctype html>
<html ng-app>
   <head>
     <script src="https://ajax.googleapis.com/ajax/libs
       /angularjs/1.5.5/angular.min.js"></script>
    </head>
    <body>
      <div>
         <label>Name: </label>
            <input type="text" ng-model="yourName">
            <hr>
            <h3>Hello {{yourName}}!</h3>
        </div>
    </body>
</html>
```

Zápis, ako je možné vidieť, je veľmi intuitívny. Vďaka nastavenia atribútu `ng-model` sa na stránke v texte „Hello ... !“ prejaví okamžite text, ktorý užívateľ napíše do vstupného poľa. Na obrázku 2.4 je možné vidieť, ako sa tento HTML kód prejaví v internetovom prehliadači.



Obr. 2.4: Jednoduchá ukážka použitia Angularu

Aktuálne stabilná verzia frameworku Angular je 1.5.5, ale v súčasnosti existuje beta verzia Angularu 2. Podľa tvorcov frameworku je možné túto verziu aj napriek beta verzii bezpečne používať.[5] Verzia Angular 2 so sebou priniesla mnohé zmeny. Prvá veľká zmena je syntax, verzia Angular 2 používá jazyk TypeScript. Podporované sú ale aj jazyky JavaScript a Dart. Oproti verzii 1.x je Angular optimalizovanejší, a tak výpočetne menej náročný a rýchlejší. V novej verzii sa dáta zobrazujú definovaním komponentov. Názorný príklad použitia viď nasledujúce štyri ukážky kódu.

[5] http://angularjs.blogspot.cz/2015/12/angular-2-beta.html

```
<!doctype html>
<html>
...
   <body>
      <hello>Loading data...</hello>
    </body>
</html>
```

Na prvej ukážke kódu sa nachádza obsah súboru `index.html`, kde je možné vidieť vlastnú nadefinovanú značku `<hello>`. Text v rámci tejto značky sa na stránke zobrazí, kým sa nenačíta komponent s obsahom dát, ktoré budú zobrazené.

```
import {bootstrap} from 'angular2/platform/browser';
import {Hello} from 'component/Hello';

bootstrap(Hello);
```

Druhá ukážka kódu predstavuje súbor `app/main.js`, kde je ako prvá importovaná funkcia `bootstrap`, ktorá zaistí spustenie Angularu. Ďalej je importovaný hlavný komponent `Hello` a nakoniec je spustený Angular pomocou naimportovanej funkcie `bootstrap` s komponentom `Hello`.

```
import {Component} from 'angular2/core';

@Component({
   selector: 'hello',
   templateUrl: 'app/hello.html'
})
export class Hello {
   name: string = '';
}
```

V tretej ukážke kódu je obsah súboru `app/hello.js`, kde je možné vidieť samotné definovanie komponenty. Na začiatku je importovaný dekorátor (angl. decorator) `Component`, ktorý je následne použitý. Komponentu je nastavený parameter selektor a URL adresa (jednotný vyhľadávač zdrojov) k šablóne. Nasleduje definovanie triedy `Hello`, kľúčové slovo `export` umožňuje triedu používať v iných súboroch (v tomto prípade je importovaná v súbore `app/hello.js`).

```
<label>Name:</label>
<input type="text" [(ngModel)]="name">
<hr>
<h3>Hello {{name}}!</h3>
```

V poslednej ukážke kódu je obsah súboru `app/hello.html`, ktorý prepíše text „Loading data...“ zo súboru `index.html` akonáhle bude vytvorený komponent. Hranaté zátvorky pri atribúte `ngModel` v elemente `input` znamenajú synchronizáciu dát smerom z komponentu do DOM. Naopak, guľaté zátvorky určujú synchronizáciu dát z DOM do komponentu. Pre zápis synchronizácie dát z komponentu do DOM v texte medzi elementmi slúžia dve zložené zátvorky za sebou. Výstup bude presne taký istý ako na obrázku 2.4.

Framework Angular používa veľké množstvo webových stránok, ako napríklad PayPal,

Freelancer a The Weather Channel[6].

2.2 Backbone.js

Backbone.js[7] je knižnica, ktorá bola vytvorená v roku 2010 a jej autor je Jeremy Ashkenas [13]. Backbone je knižnica, ktorá by mala tvoriť základ celej JavaScriptovej časti aplikácie. Knižnica je závislá na jednej JavaScriptovej knižnici, a to Underscore.js[8]. Pri používaní knižnice Backbone pracuje programátor so základnými objektami:

- **Model** (Model) - Modely sú základom každej JavaScriptovej aplikácie, ktorá obsahuje interaktívne dáta a logiku, ktorá tieto dáta obklopuje. Dáta sú reprezentované ako modely a môžu byť vytvorené, validované, zničené a uložené na server. Model je možné rozšíriť vlastnými operáciami s ním (metódy) a vlastnosťami (atribúty). Knižnica Backbone poskytuje veľké množstvo prostriedkov pre prácu s modelmi, napr. inicializácia modelu, vymazanie všetkých atribútov modelu, uloženie modelu do databázy atď.

- **Udalosť** (Event) - Udalosť je tiež model s možnosťou pridať sa k inému objektu, a tak umožniť objektu viazať a spúšťať vlastné udalosti. Udalosti nemusia byť deklarované predtým než sú naviazané a môžu niesť argumenty.

- **Kolekcia** (Collection) - Kolekcia je usporiadaná množina modelov. Kolekcia typicky obsahuje modely toho istého typu, ale jednotlivé modely môžu byť súčasťou rôznych kolekcií. Knižnica poskytuje množstvo metód pre prácu s kolekciami a zároveň je možné použiť mnohé metódy, ktoré poskytuje knižnica Underscore.js, napr. získanie počtu modelov v kolekcii, klonovanie kolekcie atď.

- **Smerovač** (Router) - Webové aplikácie často poskytujú URL adresy určujúce dôležité miesta v aplikácii. Od príchodu History API je možné použiť štandardné URL (`/page`). Smerovač poskytuje metódy pre smerovanie klientskych webových stránok a spájanie ich s akciami (napr. stlačenie tlačítka) a udalosťami.

- **Synchronizácia** (Sync) - Synchronizácia predstavuje funkciu Backbone, ktorá sa volá vždy pri načítavaní modelu zo servera alebo pri ukladaní modelu na server. Východisková implementácia používa knižnicu jQuery[9] pre vytvorenie asynchrónneho požiadavku na server. Túto implementáciu je ale možné prepísať vlastným riešením.

- **Pohľad** (View) - Pohľady umožňujú členiť aplikáciu na menšie časti. Pri zmene dát je prekreslená iba odpovedajúca časť a nie je nutné prekresľovať celú stránku. Backbone neposkytuje žiadny vlastný šablónovací nástroj. Je možné ho využiť s ľubovoľnou JavaScriptovou knižnicou, napr. Handlebars[10] alebo Underscore Templates.

Knižnicu Backbone využívajú mnohé webové projekty, ako napríklad Airbnb, Disqus a SoundCloud[11].

[6] https://www.paypal.com/, https://www.freelancer.com/, http://www.weather.com/
[7] http://backbonejs.org/
[8] http://underscorejs.org/
[9] https://jquery.com/
[10] http://handlebarsjs.com/
[11] https://www.airbnb.com/, https://disqus.com/, https://soundcloud.com/

2.3 Ember.js

Framework Ember.js[12] určený pre tvorbu webových aplikácií existuje od roku 2011, kedy bol premenovaný z pôvodného názvu SproutCore 2.0 na aktuálny názov Ember [3]. Jeho autorom je Yehuda Katz. Ember využíva architektonický vzor MVC a komunikuje cez REST rozhranie. Na rozdiel od niektorých iných rámcov, Ember poskytuje celkové riešenie aplikačných súčastí na klientskej strane. Framework tiež poskytuje obojsmernú synchronizáciu dát a vkladanie závislostí.

Základnými stavebnými kameňmi tohto frameworku sú:

- **Cesta** (Route) - Každý stav aplikácie je reprezentovaný pomocou URL adresy. Každá URL adresa má odpovedajúci Route objekt, ktorý kontroluje, čo je užívateľovi viditeľné.
- **Šablóna** (Template) - Ember používa šablónovaciu knižnicu Handlebars pre vytváranie užívateľského rozhrania aplikácie. Handlebars šablóny obsahujú statické HTML a dynamický obsah vnútri Handlebars výrazov, ktorý je vyvolaný použitím dvoch párov zložených zátvoriek .
- **Komponent** (Component) - Komponenty slúžia k vytváraniu vlastných HTML značiek, ktoré je potom možné použiť v šablónach. Komponent pozostáva z JavaScriptovej časti, ktorá určuje chovanie a odpovedajúca Handlebars šablóna. Komponenty majú hlavne vizuálnu funkciu, nevedia o ich obklopujúcom kontexte, ale môžu mať vstup a môžu posielať správy smerujúce k Route.
- **Model** (Model) - Každá cesta má priradený objekt Model, ktorý zahŕňa dáta spojené s aktuálnym stavom aplikácie. Pre načítanie JSON[13] objektov zo servera a použitie týchto objektov ako modely je možné použiť jQuery[14], väčšina aplikácií však pre tento účel používa knižnicu Ember Data.

Ember využívajú mnohé webové stránky, ako napríklad Discourse, Vine a Chipotle[15]. Aj napriek tomu, že je tento framework zameraný najmä na webové aplikácie, je možné pomocou neho vytvoriť aj desktopové alebo mobilné aplikácie. Príkladom desktopovej aplikácie využívajúcej framework Ember je Apple Music[16].

2.4 React

React[17] je pomerne nový framework, ktorý založil Jordan Walke v roku 2013 a aktuálne ho vyvíja spoločnosť Facebook [4]. Tento framework je zameraný čisto na tvorbu užívateľských rozhraní. Od ostatných rámcov sa líši hlavne tým, že vkladá HTML kód do JavaScriptového kódu. Hlavnými črtami tohto frameworku sú:

- React používa jednosmerný tok dát, na ktorý je možné nahliadať ako na jednosmernú synchronizáciu dát. Dáta sa pohybujú smerom od rodičov k potomkom. Podľa autorov frameworku dôvodom tejto funkcionality je fakt, že vo Von Neumannovej architektúre

[12] http://emberjs.com/
[13] http://www.json.org/
[14] https://jquery.com/
[15] https://www.discourse.org/, https://vine.co/, https://www.chipotle.com/
[16] http://www.apple.com/music/
[17] https://facebook.github.io/react/

je tok dát tiež jednosmerný.[18] Framework ale poskytuje prostriedky, pomocou ktorých je možné dosiahnuť obojsmerný tok dát.

- Framework používa virtuálny DOM. React vkladá HTML kód do JavaScriptvého kódu, a tým v podstate užívateľa odtieni od objektového modelu dokumentu. Framework si teda vytvorí vlastný virtuálny DOM, ktorý porovnáva so skutočným objektovým modelom dokumentu, nájde rozdiely a efektívne ho aktualizuje.
- Aby bolo možné používať HTML v JavaScriptovom kóde tak ako sú programátori zvyknutí, autori vytvorili jazyk JSX[19], nadstavbu nad JavaScriptom.
- V roku 2015 bol predstavený React Native[20], ktorý umožní vývoj natívnej mobilnej aplikácie pre iOS a Android v JavaScripte a v niektorých vybraných častiach CSS.

Knižnicu React používajú mnohé webové stránky ako napríklad Imgur, Bleacher Report a Feedly[21].

2.5 Knouckout.js

Framework Knouckout.js[22] vznikol v roku 2010 a jeho autorom je Steve Sanderson [10]. Na rozdiel od predchádzajúcich uvedených frameworkov Knockout používa architektonický vzor MVVM, derivát návrhovej paradigmy MVC. Na rozdiel od MVC tvorí časť ViewModel spojovaciu vrstvu medzi časťami View a Model. Výhodou tohto frameworku je, že môže byť pridaný do existujúcej webovej aplikácie bez výraznejších zmien v architektúre. Základné koncepty frameworku Knockout:

- **Sledovanie závislostí** - automatické obnovenie správnych častí užívateľského rozhrania kedykoľvek nastane zmena dátového modelu.
- **Deklaratívna synchronizácia** - jednoduchý spôsob prepojenia jednotlivých častí užívateľského rozhrania a dátového modelu. Je možné vytvoriť komplexné, dynamické užívateľské rozhrania použitím ľubovoľne zanorených previazaných kontextov.
- **Jednoduchosť rozšíriteľnosti** - implementácia vlastnej funkcionality ako nového deklaratívneho viazania pre jednoduché znovu použitie.

Knockout.js používajú napríklad webové stránky portálu Windows Azure, JSFiddle a AMC Theatres[23].

[18] https://facebook.github.io/react/docs/two-way-binding-helpers.html
[19] https://facebook.github.io/react/docs/jsx-in-depth.html
[20] https://facebook.github.io/react-native/
[21] http://imgur.com/, http://bleacherreport.com/, https://feedly.com/
[22] http://knockoutjs.com/
[23] http://manage.windowsazure.com/, https://jsfiddle.net/, https://www.amctheatres.com/

Kapitola 3

Serverová časť webovej aplikácie

V úvode tejto kapitoly je stručne popísaný programovací jazyk Java, architektonický štýl Representational State Transfer a platforma Java so zameraním na tvorbu serverovej časti webovej aplikácie. V podkapitole 3.1 je možné vidieť stručný popis a ukážku rozhrania pre programovanie aplikácií (angl. Application Programing Interface (API)) JAX-RS[1] a v podkapitole 3.2 popis a ukážku použitia frameworku Spring[2].

Pôvodne sa programovací jazyk **Java** [11] volal Oak a vznikol v roku 1991. Jeden z hlavných tvorcov je James Gosling a cieľom bolo vytvorenie programovacieho jazyka pre spotrebnú elektroniku. Programovací jazyk Oak bol premenovaný na Java a v dobe rozmachu Internetu bolo jeho zameranie zmenené zo spotrebnej elektroniky na web.

Java je objektovo orientovaný programovací jazyk. Je to aj platforma, ktorú je možné použiť na tvorbu webových aplikácií. Rovnako môže byť Java použitá na tvorbu desktopových aplikácií alebo mobilných aplikácií. Existuje niekoľko balíkov programovacieho jazyka Java:

- **Java ME** (Micro Edition) - pre mobilné telefóny a malé zariadenia,
- **Java SE** (Standard Edition) - inštalácia Javy pre domáce počítače,
- **Java EE** (Enterprise Edition) - používaná v enterprise sektore,
- **Java Card** - pre implementáciu do inteligentných čipových kariet,
- niektoré ďalšie pre špecifické úlohy.

Representational State Transfer [7] (REST) je architektonický štýl aplikácií typu klient-server, ktorý sa sústreďuje na prenos reprezentácií zdrojov prostredníctvom požiadaviek a odpovedí. Dáta a funkcionalita sú v tomto architektonickom štýle považované za zdroje a sú prístupné použitím jednotnej identifikácie zdroja (angl. Uniform Resourse Identifier (URI)), typicky odkazmi na web. So zdrojmi sa pracuje ako s dokumentami pomocou niekoľkých jednoduchých operácií.

REST zdroj môže byť napríklad aktuálny stav počasia v určitom meste. Reprezentácia tohto zdroja môže byť XML dokument, obrázok alebo HTML stránka. Klient môže obdržať určitú reprezentáciu, aktualizovať zdroj dátami z reprezentácie alebo môže vymazať zdroj úplne.

[1] https://jax-rs-spec.java.net/
[2] https://spring.io/

Tento architektonický štýl je navrhnutý na používanie bezstavového komunikačného protokolu, typicky HTTP. Klienti a servery si vymieňajú reprezentácie zdrojov použitím štandardizovaného rozhrania alebo protokolu.

REST webové služby sú teda voľne viazané a ľahké webové služby, ktoré sú obzvlášť vhodné pre vytvorenie rozhrania pre programovanie aplikácií pre klientov.

Tieto nasledujúce princípy napomáhajú jednoduchosti, ľahkosti a rýchlosti REST aplikácií:

- **Identifikovanie zdroja pomocou URI**: REST webové služby vystavujú množinu zdrojov, ktoré definujú oblasti, v ktorých klient komunikuje so zdrojmi. Zdroje sú identifikované pomocou URI, ktoré poskytuje globálny adresový priestor.
- **Jednotné rozhranie**: Zdroje sú manipulované iba pomocou štyroch operácií - vytvorenie, získanie, obnovenie a vymazanie. Operácia vytvorenia (PUT) vytvorí nový zdroj, ten môže byť odstránený operáciou vymazania (DELETE). Operácia získania (GET) obdrží aktuálny stav zdroja v nejakej reprezentácii a operácia obnovenia (POST) odošle nový stav na zdroj.
- **Seba popisujúce správy**: Zdroje sú oddelené od svojich reprezentácií, a tak ich obsah môže byť dostupný v rôznych formátoch, napríklad HTML, XML, čistý text, PDF, JPEG, JSON a iné. Metadáta zdrojov sú prístupné a môžu byť použité napríklad na riadenie vyrovnávacej pamäti.

3.1 JAX-RS

JAX-RS je API programovacieho jazyka Java navrhnuté na jednoduché vytvorenie aplikácií používajúcich REST architektúru [8]. JAX-RS API používa anotácie programovacieho jazyka Java pre zjednodušenie vývoja REST webových služieb. Programátori dopĺňajú Java triedy JAX-RS anotáciami, ktoré definujú zdroje a akcie, ktoré sa majú byť vykonané na týchto zdrojoch. JAX-RS anotácie sú spracovávané za behu behovým prostredím.

V tabuľke 3.1 sú uvedené niektoré anotácie programovacieho jazyka Java spolu so stručným popisom ich použitia.

Na nasledujúcej ukážke kódu je možné vidieť jednoduchý príklad zdrojovej triedy, ktorá používa JAX-RS anotácie.

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

@Path("/helloworld")
public class HelloWorld {

   @GET
   @Produces("text/plain")
   public String getMessage() {
      return "Hello World!";
   }
}
```

Na začiatku je nutné požadované anotácie importovať. Následne sú anotácie spracovávané za behu programu. Anotácia `@Path` špecifikuje URI adresu zdroja, ktorá je „`/helloworld`“.

Anotácia	Popis
`@Path`	Hodnotou anotácie `@Path` je relatívna URI adresa, ktorá určuje adresu zdroja, napr. `/hello`.
`@GET`	Anotácia `@GET` určuje typ požiadavky a odpovedá HTTP metóde GET. Metóda s priradenou anotáciou `@GET` vykoná HTTP GET požiadavku. Chovanie zdroja sa potom odvíja od HTTP metódy, na ktorú zdroj reaguje.
`@POST`	Anotácia `@POST` určuje typ požiadavky a odpovedá HTTP metóde POST. Metóda s priradenou anotáciou `@POST` vykoná HTTP POST požiadavku. Chovanie zdroja sa potom odvíja od HTTP metódy, na ktorú zdroj reaguje.
`@PUT`	Anotácia `@PUT` určuje typ požiadavky a odpovedá HTTP metóde PUT. Metóda s priradenou anotáciou `@PUT` vykoná HTTP PUT požiadavku. Chovanie zdroja sa potom odvíja od HTTP metódy, na ktorú zdroj reaguje.
`@DELETE`	Anotácia `@DELETE` určuje typ požiadavky a odpovedá HTTP metóde DELETE. Metóda s priradenou anotáciou `@DELETE` vykoná HTTP DELETE požiadavku. Chovanie zdroja sa potom odvíja od HTTP metódy, na ktorú zdroj reaguje.
`@HEAD`	Anotácia `@HEAD` určuje typ požiadavky a odpovedá HTTP metóde HEAD. Metóda s priradenou anotáciou `@HEAD` vykoná HTTP HEAD požiadavku. Chovanie zdroja sa potom odvíja od HTTP metódy, na ktorú zdroj reaguje.
`@Consumes`	Anotácia `@Consumes` je používaná pre špecifikáciu MIME typu internetového média, ktorý zdroj dokáže prijať a je odoslaný klientom.
`@Produces`	Anotácia `@Produces` je používaná pre špecifikáciu MIME typu internetového média, ktorý dokáže zdroj produkovať a poslať naspäť klientovi, napr. `"text/plain"`.
`@ApplicationPath`	Anotácia `@AplicationPath` sa používa pre definovanie URL mapovania aplikácie. Cesta je špecifikovaná prostredníctvom `@AplicationPath`, čo predstavuje základ adresy URI pre všetky URI adresy zdrojov špecifikovaných pomocou anotácie `@Path`.

Tabuľka 3.1: JAX-RS anotácie

Anotácia `@GET` špecifikuje, že metóda vykoná HTTP `GET` požiadavku. Nakoniec anotácia `@Produces` udáva, že metóda bude produkovať obsah identifikovaný viacúčelovým rozšírením internetovej pošty (angl. Multipurpose Internet Mail Extensions (MIME)) typom média `text/plain`.

3.2 Spring

Spring je framework pre vývoj Java EE aplikácií [9]. Framework môže byť použitý ľubovoľnou Java aplikáciou. Je populárne jeho použitie ako alternatíva k Enterprise Java Beans (EJB) alebo ako jeho nadstavba. Spring sa skladá z častí organizovaných do modulov. Tieto moduly sú rozdelené do vrstiev, napr.:

- **Core Container** - vrstva obsahujúca základné moduly frameworku.

- **Aspect-oriented programming** - modul implementujúci podporu pre aspektovo orientované programovanie.
- **Data Access/Integration** - vrstva zabezpečuje podporu práce s databázou, mapovanie medzi Java objektami a XML dokumentmi a iné.
- **Web** - vrstva obsahujúca moduly podporujúce webové technológie.
- **Test** - umožňuje používanie a písanie unit testov.

Vrstva Web zahŕňa modul `spring-webmvc`, ktorý obsahuje MVC framework a implementáciu REST webových služieb pre tvorbu webových aplikácií. MVC framework umožňuje oddelenie doménového modelu od webovej formy a spája všetky ostatné možnosti poskytované frameworkom Spring. Na nasledujúcej ukážke kódu je možné vidieť jednoduchý príklad použitia frameworku Spring.

```
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@RestController
@RequestMapping("/helloworld")
public class HelloWorld {

   @RequestMapping(method = RequestMethod.GET)
   public String printHello() {
      return "Hello World!";
   }
}
```

Triedy sú dopĺňané anotáciami. Tieto anotácie definujú zdroje a akcie, ktoré sa na nich majú vykonať. Spracovanie anotácií prebieha aj v tomto prípade za behu programu. HTTP metódy `GET`, `POST`, `PUT` a `DELETE` je možné vykonať použitím anotácie `@RequestMapping`. Táto anotácia má atribút `method`, ktorý môže nadobúdať hodnoty odpovedajúce HTTP metódam.

Kapitola 4

FITLayout

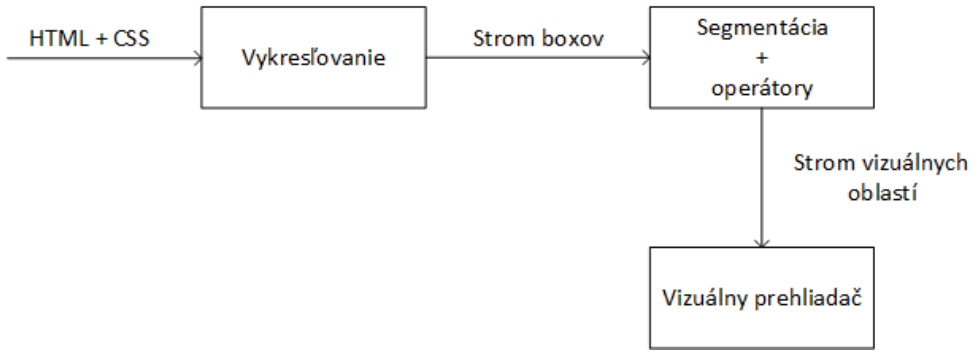
FITLayout [1] je rozšíriteľný nástroj na segmentáciu webových stránok a analytický framework napísaný v programovacom jazyku Java. Definuje obecné Java rozhranie pre reprezentáciu a vykreslenie webovej stránky a jej rozdelenie na oblasti pre ich ďalšiu analýzu. Poskytuje nástroje pre implementáciu segmentačných algoritmov so spoločným aplikačným rozhraním. Následne je možné na výsledok segmentácie pomocou ďalších nástrojov použiť ďalšie textové alebo vizuálne klasifikačné metódy. FITLayout ešte poskytuje nástroje pre ovládanie procesu segmentácie a skúmania výsledkov segmentácie pomocou grafického užívateľského rozhrania.

4.1 Architektúra

FITLayout pracuje na vykreslenej stránke reprezentovanou stromom boxov. Strom boxov je získaný vykreslením stránky a vypočítaním pozícií, písma, farby a iných vizuálnych vlastností boxov. Strom boxov je potom vstupom pre segmentačné algoritmy.

Nástroj analyzuje vstupný strom boxov a vytvorí strom vizuálnych oblastí, ktoré odpovedajú detekovaným vizuálnym oblastiam na stránke. Tak je vytvorený základný strom vizuálnych oblastí. Na tento základný vytvorený strom vizuálnych oblastí môžu byť použité ďalšie operátory. Použitím týchto operátorov sa tak môže zmeniť štruktúra výsledného vizuálneho stromu oblastí.

Proces vykresľovania stránky a segmentácie môže byť ovládaný použitím skupiny poskytnutých nástrojov. Tieto nástroje zahŕňajú vizuálny prehliadač s grafickým užívateľským rozhraním. Toto rozhranie môže byť použité pre konfiguráciu a vykonanie jednotlivých úloh. Skriptovateľný procesor umožňuje použiť JavaScript pre beh úloh v dávkovom režime.



Obr. 4.1: Architektúra FITLayout

FITLayout framework pozostáva z týchto základných modulov:

- **API** - základné rozhrania a ich základná implementácia, ktorá definuje spoločné aplikačné rozhranie pre segmentačné metódy stránky. API poskytuje spoločné rozhranie, ktoré spája všetky ostatné moduly.
- **CSSBox väzba** - východisková implementácia vykreslenej stránky založená na vykresľovacom stroji CSSBox[1].
- **Segmentácia** - implementácia základnej segmentačnej metódy, ktorá môže byť rozšírená vlastnými operátormi.
- **Klasifikácia** - implementácia iných textových alebo vizuálnych klasifikačných metód, ktoré môžu byť použité pre automatické označenie oblastí založených na ich vizuálnych alebo textových vlastnostiach.
- **Nástroje** - nástroje pre ovládanie segmentácie, ktoré zahŕňajú grafický prehliadač výsledku segmentácie.

Architektúra nástroja je ľahko rozšíriteľná vytvorením nových zásuvných modulov. Tieto moduly môžu poskytovať novú funkcionalitu ako nový vykresľovač dokumentov, segmentačný algoritmus, operátory na ďalšie spracovanie stromu vizuálnych oblastí alebo rozšírenie grafického užívateľského rozhrania.

FITLayout umožňuje vytvárať tieto druhy služieb:

- `BoxTreeProvider` - vykresľovač stránok založený na vstupných parametroch, napr. URL stránky. Služba vykreslí stránku a vytvorí strom boxov.
- `AreaTreeProvider` - východiskový segmentačný algoritmus. Na vstupe má strom boxov a na výstupe strom vizuálnych oblastí, ktorý reprezentuje segmentovanú stránku.
- `AreaTreeOperator` - ďalšie operátory, ktoré je možno aplikovať na strom vizuálnych oblastí. Môže mierne modifikovať strom, ako napríklad spojiť alebo rozdeliť uzly.
- `LogicalTreeProvider` - analyzátor, ktorý má na výstupe výsledný strom vizuálnych oblastí, ktorý priradí význam vybraným oblastiam.

Každá služba je identifikovaná unikátnym identifikátorom. Všetky služby môžu prijímať vstupné parametre. Implementujú rozhranie, ktoré umožňuje získať informácie o vyžadovaných vstupných parametroch a priradiť im hodnotu.

4.2 Strom boxov

Celá vykreslená stránka je reprezentovaná objektom. Je získaný koreňový uzol, ktorý reprezentuje obsah stránky. Uzly stromu boxov sú reprezentované individuálnymi vykreslenými boxami. Každý tento box má fixnú pozíciu na vykreslenej stránke a ďalšie vlastnosti, napr. písmo, veľkosť písma, farba atď. Boxy môžu byť jedným z nasledujúcich typov:

- `ELEMENT` - box reprezentovaný DOM elementom.
- `TEXT_CONTENT` - box reprezentovaný zobrazeným textom.

[1] http://cssbox.sourceforge.net/documentation.php

- `REPLACED_CONTENT` - box reprezentovaný obrázkom alebo inými objektmi.

Boxy sú organizované do hierarchickej štruktúry, ktorú je možné prechádzať. Boxy `TEXT_CONTENT` a `REPLACED_CONTENT` vždy tvoria listové uzly stromu a `ELEMENT` sa môže nachádzať v hociktorej časti stromu.

Východisková implementácia vytvorenia stromu boxov, tj. vykreslenia stránky je založená na stroji CSSBox. Vstupné dokumenty sú identifikované pomocou URL adresy a sú podporované formáty HTML/CSS a PDF.

4.3 Segmentácia

Segmentačný algoritmus má na vstupe strom boxov a na výstupe strom vizuálnych oblastí. Výsledný strom je reprezentovaný objektom. Je získaný koreň stromu vizuálnych oblastí, ktorý odpovedá výsledku segmentácie. Každý uzol vizuálneho stromu oblastí je reprezentovaný objektom a odpovedá vizuálnej oblasti detekovanej na stránke. Koreňový uzol reprezentuje oblasť celej stránky a potomkovia rodičovských uzlov potom tvoria menšie detekované oblasti. Listy stromu môžu byť tvorené boxami zo stromu boxov, ktoré predstavujú obsah oblasti. Cez uzly stromu je možné prechádzať a manipulovať obdobne ako v prípade strom boxov.

Pozícia oblasti vo vykreslenej stránke a jej vizuálnych vlastností ako písmo, farba sú implementované podobne ako v prípade stromu boxov. Keďže obsiahnuté boxy môžu mať rôzne vizuálne vlastnosti, odpovedajúce metódy pre vizuálne oblasti vrátia priemernú hodnotu jednotlivých hodnôt z celej oblasti.

4.4 Nástroje

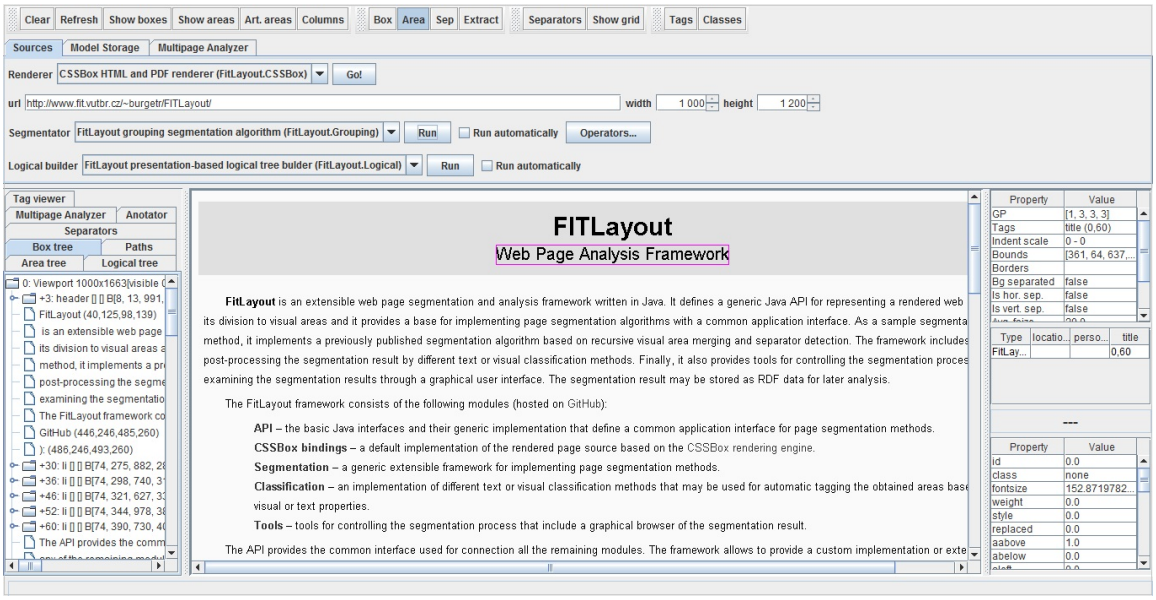
Tento modul poskytuje nástroje `Processor` a `BlockBrowser` pre beh a kontrolu procesu segmentácie. `Processor` implementuje celú segmentáciu, napr. vytvorenie stromu základných vizuálnych oblastí a aplikovanie operátorov na tento strom. Sú k dispozícii dve implementácie procesora:

- `ScriptableProcessor` používa JavaScript pre nastavenia operátorov, ktoré majú byť použité.
- `GUIProcessor` - nastavenia operátorov môžu byť zmenené v grafickom užívateľskom rozhraní.

`BlockBrowser` implementuje východiskový prehliadač pomocou knižnice Swing GUI.

4.5 Rozhranie aplikácie

Na obrázku 4.2 je možné vidieť súčasnú podobu desktopovej aplikácie nástroja FITLayout. Po spustení nástroja FITLayout je možné vybrať vykresľovač (Renderer), segmentátor (Segmentator), logický analyzátor (Logical Builder), vybrať operátory, ktoré budú použité na základný vizuálny stom. Do vstupného riadka je možné zadať ľubovoľnú URL adresu a nastaviť ľubovoľnú šírku a výšku vykreslenej stránky. V ľavej časti okna je možné prechádzať štruktúru stromu boxov, stromu vizuálnych oblastí alebo stromu logických oblastí. V pravej časti okna sú vypísané vizuálne vlastnosti oblastí a ich hodnoty. V hlavnej časti okna je vykreslený dokument. Po kliknutí na uzol v strome boxov, strome vizuálnych oblastí alebo strome logických oblastí je príslušný box, vizuálna oblasť alebo logická oblasť vyznačená vo vykreslenom dokumente v hlavnej časti okna.



Obr. 4.2: FITLayout

4.6 Súčasný stav a plán práce

V súčasnosti existuje desktopová verzia nástroja FITLayout. Z pohľadu dnešnej doby sú desktopové aplikácie stále viac zatláčané do úzadia a sú nahradzované webovými aplikáciami. Dôvodov je hneď niekoľko, pravdepodobne najväčšou výhodou je dostupnosť webovej aplikácie z webového prehliadača bez nutnosti inštalácie špeciálneho softwaru. Podobne je veľkou výhodou nezávislosť na operačnom systéme, čo často znamená zníženie nákladov na tvorbu niekoľkých verzií desktopovej aplikácie pre jednotlivé operačné systémy. Nevýhodou je naopak veľká závislosť na poskytovateľovi aplikácie. Ak sa poskytovateľ rozhodne ukončiť poskytovanie webovej služby, nie je možné ju ďalej používať, na rozdiel od lokálne inštalovaného softwaru. Ďalšou nevýhodou je tiež situácia, kedy dôjde k prerušeniu spojenia so serverom poskytovateľa. Počas tejto doby môže byť webová aplikácia nedostupná. Aj napriek existujúcim nevýhodám zjavne prevažujú výhody webových aplikácií a ich popularita stále stúpa. Preto je veľmi vhodné zamerať sa na vytvorenie webovej aplikácie, ktorá bude ponúkať obdobné funkcie ako súčasné grafické nástroje dostupné vo nástroji FITLayout.

Ďalším postupom je vytvoriť webovú aplikáciu, ktorá bude jednoducho a intuitívne ovládateľná a bude poskytovať hlavné funkcie, ktoré ponúka nástroj FITLayout. Pôjde hlavne o tieto funkcie:

- možnosť zadania ľubovoľnej URL adresy dokumentu, ktorá bude analyzovaná,
- možnosť vybrať vykresľovač zo zoznamu prístupných vykresľovačov a ten potom použiť pri vykresľovaní,
- možnosť vybrať segmentátor zo zoznamu prístupných segmentátorov a ten použiť pri segmentácii dokumentu,
- možnosť vybrať operátory, ktoré budú použité pri segmentácii dokumentu,
- možnosť vybrať logický analyzátor zo zoznamu prístupných logických analyzátorov a ten použiť pri tvorbe stromu logických oblastí,

- možnosť zadať požadovanú šírku a výšku vykresleného dokumentu,
- zobrazenie stromovej štruktúry stromu boxov, stromu vizuálnych oblastí a stromu logických oblastí,
- vykreslenie analyzovaného dokumentu vybraným vykresľovačom,
- možnosť vyberať uzly v stromovej štruktúre a tak zároveň zvýrazniť príslušnú oblasť vo vykreslenom dokumente,
- výpis vizuálnych vlastností jednotlivých boxov a ich hodnôt.

Kapitola 5

Návrh aplikácie

Obsahom tejto kapitoly je popis návrhu klientskej aplikácie, ktorá bude implementovaná. V podkapitole 5.1 sa nachádza návrh a popis grafického užívateľského rozhrania, v podkapitole 5.2 je načrtnutá architektúra nástroja a v podkapitolách 5.3 a 5.4 sa nachádza stručný popis návrhu klienta a servera vrátane služieb, ktoré bude server poskytovať

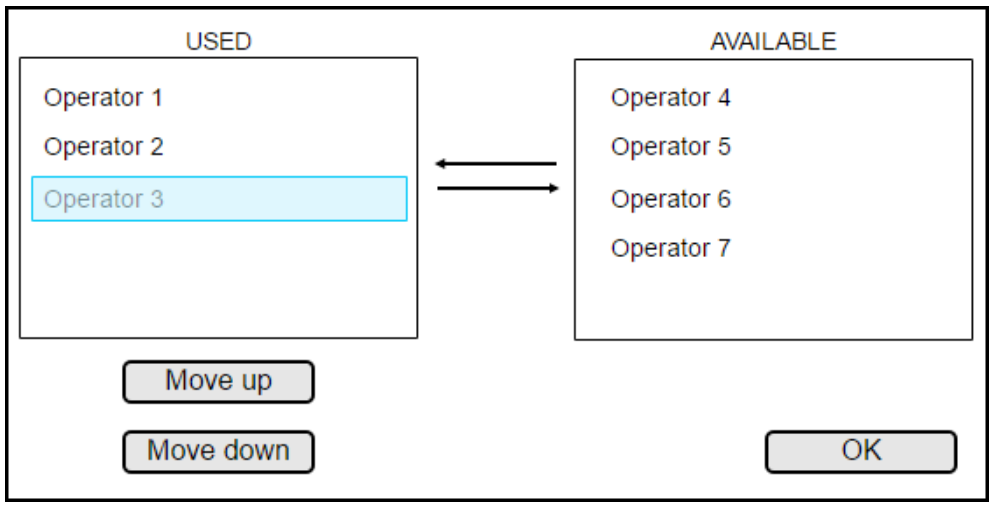
5.1 Návrh grafického užívateľského rozhrania

Výsledný nástroj s webovým rozhraním bude zachovávať hlavnú koncepciu grafického užívateľského rozhrania aktuálnej desktopovej verzie. Jedinou výraznejšou zmenou bude skrytie možností nastavenia a operátorov do samostatného modálneho okna, aby zaberali čo najmenej miesta v hlavnom okne. V hornej časti okna bude teda možnosť výberu vykresľovača a ostatných nástrojov, v ľavej časti stránky bude stromová štruktúra stromu, v strednej časti vykreslený dokument a nakoniec v pravej časti okna vizuálne vlastnosti oblastí a ich hodnoty. Náčrt grafického užívateľského prostredia viď obrázok 5.1.

Obr. 5.1: Návrh grafického užívateľského rozhrania

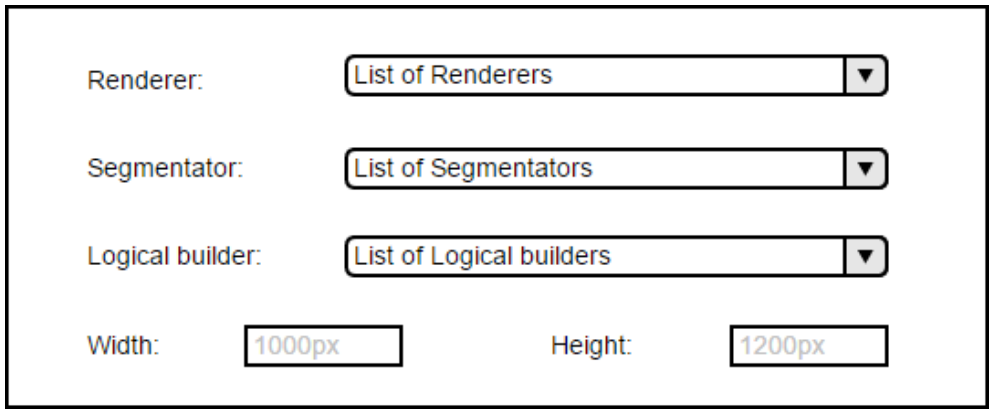
V hornej časti stránky bude možnosť zadania URL adresy dokumentu, po kliknutí na tlačítko Segment bude spustená segmentácia dokumentu. Stromová štruktúra bude zobrazená v ľavej časti okna. Táto časť tiež bude obsahovať záložky na prehliadanie stromovej štruktúry stromu boxov, stromu vizuálnych oblastí a stromu logických oblastí. V najväčšej, strednej časti obrazovky bude vykreslený požadovaný dokument vybraným vykresľovačom. V pravej časti budú vypísané vizuálne vlastnosti vybranej oblasti alebo uzlu a ich hodnoty.

Po kliknutí na tlačítko Operators bude otvorené modálne okno, kde bude možné nastaviť operátory, ktoré budú použité na základný vizuálny strom pri segmentácii, ich poradie a ich prípadné parametre, viď obrázok 5.2.



Obr. 5.2: Návrh grafického užívateľského rozhrania - operátory

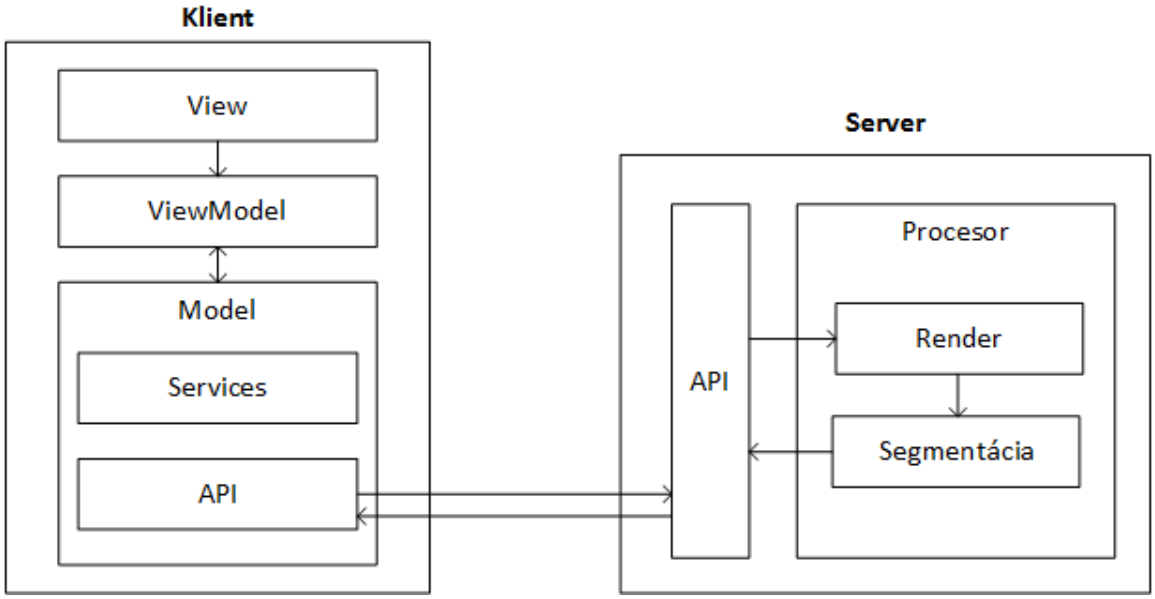
Posledným prvkom návrhu grafického užívateľského rozhrania je ikona ozubeného kolieska. Bude slúžiť na nastavenie nástrojov, ktoré budú pri segmentácii použité, viď obrázok 5.3. Konkrétne bude možné vybrať zo zoznamu vykresľovačov, segmentátorov a logických analyzátorov. Tiež bude možné zadať požadované rozmery vykresľovaného dokumentu.



Obr. 5.3: Návrh grafického užívateľského rozhrania - nastavenia

5.2 Architektúra nástroja

Výsledný nástroj bude implementovaný ako webová aplikácia a bude poskytovať obdobné funkcie, ako grafické nástroje, ktoré ponúka FITLayout. Výsledný nástroj bude webová aplikácia typu klient-server. Grafické užívateľské rozhranie a GUI logika budú implementované ako JavaScriptový klient. Server bude v tomto prípade slúžiť hlavne ako dátový zdroj, bude poskytovať rozhranie k nástroju FITLayout a pomocou tohto rozhrania budú analyzované dokumenty, tj. získavané stromy boxov, stromy vizuálnych oblastí apod. Klient pošle serveru požiadavku na získanie napríklad stromu boxov a adresu dokumentu, server tento dokument spracuje, vytvorí strom boxov a pošle ho naspäť klientovi. Klient bude vytvorený pomocou JavaScriptového frameworku, konkrétne bude použitý framework Angular. Server bude implementovaný na platforme Java, bude využívať REST webové služby a bude použitá špecifikácia JAX-RS.



Obr. 5.4: Architektúra webového nástroja

5.3 Klient

Klient bude implementovaný pomocou architektonického vzoru MVVM a bude použitý framerowk Angular. Vrstva View predstavuje grafické užívateľské rozhranie a bude tvorené HTML a CSS dokumentami. Vrstva ViewModel predstavuje kontrolnú vrstvu, sleduje zmeny v grafickom užívateľskom rozhraní a od vrstvy Model prostredníctvom komunikácie so serverom získa požadované dáta. Akonáhle vrstva View zaznamená požadované dáta v kontrolnej vrstve ViewModel, aktualizuje príslušnú komponentu v grafickom užívateľskom rozhraní týmito dátami.

Vo vrstve Model je obsiahnutý komponent Services, ktorý budú tvoriť metódy určené na manipuláciu s dátami, ak bude potrebné pred zobrazením získané dáta od servera transformovať. Vrstva API predstavuje časť klienta pre REST webové služby, ktorá bude komunikovať so serverom.

5.4 Server

Server webového nástroja bude implementovaný na platforme Java a bude použitá špecifikácia JAX-RS. Server obsahuje vrstvu API, ktorá bude poskytovať REST webové služby a bude tvoriť rozhranie nástroja FITLayout. Klient a server si vymieňajú dáta prostredníctvom transportného protokolu HTTP vo formáte JSON. Server bude poskytovať nasledujúce služby:

- **GET /renderer**

  Služba renderer zistí zoznam dostupných vykresľovačov a vráti ho klientovi.

Vstup:	žiadny vstup
Výstup:	zoznam možných vykresľovačov

- **GET /segmentator**

  Služba segmentator zistí zoznam dostupných segmentátorov a vráti ho klientovi.

Vstup:	žiadny vstup
Výstup:	zoznam možných segmentátorov

- **GET /operator**

  Služba operator zistí zoznam dostupných operátorov a vráti ho klientovi.

Vstup:	žiadny vstup
Výstup:	zoznam možných operátorov

- **GET /logical**

  Služba logical zistí zoznam dostupných logických analyzátorov a vráti ho klientovi.

Vstup:	žiadny vstup
Výstup:	zoznam možných logických analyzátorov

- **POST /tree/box**

  Na základe vstupných parametrov služba s využitím zvoleného vykresľovača vykreslí dokument a výsledný strom boxov vráti klientovi.

Vstup:	• zvolený vykresľovač • URL dokumentu • šírka vykresľovanej oblasti • výška vykresľovanej oblasti
Výstup:	strom boxov

- **POST /tree/area**

  Na základe vstupných parametrov služba s využitím zvoleného segmentátora vytvorí základný strom oblastí, na ten aplikuje operátory a výsledný strom vizuálnych oblastí vráti klientovi.

Vstup:	• zvolený segmentátor • zvolené operátory • strom boxov
Výstup:	strom vizuálnych oblastí

- **POST /tree/logical**

  Na základe vstupných parametrov služba s využitím logického analyzátora vytvorí zo stromu vizuálnych oblastí strom logických oblastí a vráti ho klientovi.

Vstup:	• zvolený logický analyzátor • strom vizuálnych oblastí
Výstup:	strom logických oblastí

Kapitola 6

Implementácia

V tejto kapitole je popísaná implementácia výsledného nástroja. V podkapitole 6.1 sú popísané technológie použité pri implementácii klientskej aplikácie. V ďalšej podkapitole 6.2 je možné nájsť realizáciu implementovanej webovej aplikácie FITLayout4Web. V nasledujúcej podkapitole 6.3 sú popísané dôležité aspekty klientskej časti a v poslednej podkapitole 6.4 dôležité aspekty serverovej časti webovej aplikácie.

6.1 Použité technológie

Pri implementácii klientskej časti bol použitý JavaScriptový framework Angular, technológie HTML, CSS a Bootstrap. Pre tvorbu serverovej časti bola použitá Java EE.

HTML [2] je skratka pre značkovací jazyk Hyper Text Markup Language. Pomocou jazyka HTML je popisovaná štruktúra webových stránok a predstavuje jednu z hlavných technológií pre tvorbu internetového obsahu. Webový prehliadač načíta HTML súbor a jeho obsah vykreslí na webovú stránku. V minulosti bol jazyk HTML tiež používaný pre popis vizuálnej podoby. Časom sa pre funkciu popisu vizuálnych častí webových stránok začal používať jazyk CSS.

Základom jazyka HTML sú značky (tagy) a ich vlastnosti (atribúty). Medzi týmito značkami je uzatváraný text dokumentu. Samotná značka je tvorená svojim názvom a uhlovými zátvorkami, napr. `<p>`. Značky sú väčšinou párové, pričom počiatočná a koncová značka sú rovnaké, ale pred koncovou značkou sa nachádza lomítko, napr. `</p>`. Časť dokumentu tvorená značkami a textom medzi týmito značkami sa nazýva prvok (element) dokumentu. Prvky je možné zanorovať, teda súčasťou prvku môžu byť ďalšie vnorené prvky.

HTML dokument má pevne danú štruktúru, ktorú je nutné dodržať za účelom správneho načítania dokumentu webovým prehliadačom. Túto štruktúru tvorí deklarácia typu dokumentu, pre súčasnú verziu značka `<!DOCTYPE html>`. Uvádza sa na začiatku dokumentu. Celý dokument reprezentuje koreňový prvok `html`. Hlavička dokumentu `head` obsahuje metadáta vzťahujúce sa k dokumentu, napr. kódovanie, názov dokumentu alebo titulok. Nakoniec prvok `body` zahŕňa obsah celého dokumentu.

Existuje mnoho skupín značiek používané na rôzne účely. Medzi tieto skupiny patria napr. značky určujúce štruktúru dokumentu. Patria sem napr. značky pre odstavce `<p></p>` alebo značky pre nadpisy `<h1></h1>`. Väčšina webových prehliadačov má zabudované východiskové formátovanie týchto elementov. Ďalšou skupinou sú napr. značky určené pre prácu so zoznamami. Do tejto skupiny patria napr. značky určujúce položku zoznamu `<li></li>`, značky pre číslovaný zoznam `<ol></ol>` alebo značky pre odrážkový zoznam `<ul></ul>`.

Od roku 2014 existuje najnovšia verzia HTML5, ktorá so sebou priniesla mnoho zmien. Medzi zaujímavé nové elementy patria sémantické elementy pre záhlavie `<header>` a zápätie `<footer>`, multimediálne elementy pre audio `<audio>` a video `<video>`. Tiež boli zavedené nové kontrolné atribúty pre prácu s formulárom, ako dátum, čas alebo email.

CSS [2] alebo kaskádové štýly (angl. Cascading Style Sheets) je jazyk pre popis zobrazenia elementov v HTML, XHTML alebo XML dokumentu. CSS je navrhnuté primárne pre možnosť oddelenia obsahu dokumentu od vzhľadu dokumentu. Toto oddelenie redukuje duplicitu popisu vzhľadu v HTML dokumente a umožňuje použitie jedného CSS súboru v rámci viacerých HTML dokumentov. Odkaz na konkrétny CSS súbor sa zvyčajne nachádza v hlavičke HTML dokumentu. Okrem tohto spôsobu použitia je možné kaskádové štýly používať aj priamo na konkrétne elementy v obsahu HTML dokumentu.

Syntax CSS tvoria pravidlá. Pravidlo je tvorené selektorom a blokom deklarácií, pričom selektory je možné zoskupovať pomocou čiarok. Blok deklarácií je obalený v zložených zátvorkách. Deklaráciu tvorí identifikátor vlastnosti a za dvojbodkou hodnota tejto vlastnosti. Jednotlivé deklarácie sú oddelené bodkočiarkou. Existuje mnoho selektorov, medzi ktoré napr. patrí selektor `body`, ktorého deklarácie budú platiť v rámci elementu body, teda v rámci obsahu celého dokumentu. Ďalším príkladom selektora je `.trieda`, ktorého deklarácie budú uplatnené na všetkých elementoch, ktorých atribút `class` nadobúda hodnotu `trieda`.

Aktuálna špecifikácia je CSS3. Táto špecifikácia priniesla mnoho zaujímavých možností. Medzi ne patrí napríklad miera priehľadnosti prvku (angl. opacity), animácia elementov, zaoblené rohy HTML prvkov, tieň blokového prvku alebo textu a možnosť transformácie prvkov, ako napr. rotácia alebo zmena veľkosti prvku.

Bootstrap[1] je knižnica pre vytváranie webových stránok a webových aplikácií [14]. V kombinácii s jazykmi HTML, CSS a JavaScript umožňuje jednoducho vytvárať pomerne rozsiahle stránky bez nutnosti písania veľkého množstva kódu. Použitie Bootstrapu urýchľuje prácu hlavne vďaka kompletne graficky spracovaným interaktívnym prvkom ako sú napr. tlačítka, vyberacie zoznamy alebo menu. Avšak štýl týchto prvkov je možné v prípade potreby prepísať vlastnou implementáciou. Ďalšou výhodou použitia Bootstrapu je implicitná podpora responzívnosti zobrazenia, tj. optimalizácia obsahu stránky v závislosti na zariadení, na ktorom je stránka zobrazovaná. Táto funkcia je založená na mriežkovom systéme. Stránka je východiskovo rozdelená na dvanásť stĺpcov, ktoré je možné zlúčiť a dosiahnuť tak rôzne rozloženie stránky, napr. v pomere 3:6:3. V prípade prezerania stránky na menšom zariadení nebudú tieto bloky zobrazené vedľa seba, ale budú zobrazené vertikálne.

Pre tvorbu klientskej časti bol vybraný framework **AngularJS** vo verzii 2. Popis tohto frameworku a ukážku použitia je možné vidieť v podkapitole 2.1. Hlavné dôvody, prečo bol vybraný tento framework sú nasledujúce:

- Angular je vydaný pod otvorenou licenciou a má veľmi početnú skupinu užívateľov vrátane spoločnosti Google.
- Framwork podporuje REST rozhranie, ktoré bolo pre nástroj FITLayout4Web navrhnuté.
- Obojsmerná synchronizácia dát a tak odtienenie práce s DOM stromom.
- Deklaratívny spôsob práce s užívateľským rozhraním - rozšírenie jazyka HTML.
- Je dostupná veľmi rozsiahla a kvalitná dokumentácia.

[1] http://getbootstrap.com/

- Framework zvyšuje prehľadnosť zdrojového kódu možnosťou tvorby a práce s komponentmi.

Klientska časť bola implementovaná použitím jazyka **JavaScript**, ktorý bol stručne popísaný v úvode kapitoly 2. Bola použitá najnovšia verzia ES2015, ktorá ale staršími prehliadačmi nie je podporovaná. Z tohto dôvodu bol použitý nástroj Babel[2], pomocou ktorého je vykonávaná transformácia moderných, nepodporovaných jazykových konštrukcií tak, aby boli použiteľné aj v starších prehliadačoch. A to za pomoci starších jazykových konštrukcií.

Pre správu závislostí klientskej časti nástroja bol použitý nástroj npm[3], ktorý beží v prostredí Node.js.

Pre zostavenie klientskej časti aplikácie bol využitý nástroj Gulp[4]. Zaisťuje spúšťanie nástroja Babel, transformáciu Sass[5] na CSS a kopírovanie všetkých závislostí na jedno miesto.

Sass je jazyk, ktorý slúži pre zápis kaskádových štýlov. Rozširuje jazyk CSS o nové konštrukcie ako je zanorovanie štýlov alebo vytváranie premenných. Jazyk je kompilovaný do CSS. V aplikácii je využitý najmä kvôli knižnici Bootstrap.

Pre implementáciu serverovej časti bola použitá platforma **Java EE** (Java Platform, Enterprise Edition). Java EE je súčasťou platformy Java a je určená pre vytváranie informačných systémov a podnikových aplikácií. Účelom platformy Java EE je poskytnúť užitočnú množinu aplikačných rozhraní, ktoré šetria čas vývoja aplikácie, redukujú zložitosť aplikácie a vylepšujú výkonnosť aplikácie. Z tejto množiny aplikačných rozhraní boli využité iba niektoré časti, konkrétne Servlet API a JAX-RS. Popis API JAX-RS viď podkapitola 3.1. Celá aplikácia je zabalená v jednom výslednom balíku WAR (Web Application Archive).

Poslednou významnou použitou technológiou je **Apache Maven**[6]. Projekt je popísaný pomocou Project Object Model (POM), čo je objektová reprezentácia informácií o projekte, ako je napr. správa závislostí alebo zostavenie projektu. Hlavným účelom tohto nástroja je správa závislostí. Každá závislosť je určená unikátnym názvom a požadovanou verziou, napr. závislosť na nástroji FITLayout. Nástroj potom dokáže automaticky stiahnuť chýbajúce závislosti z centrálneho úložiska. Ďalšou funkciou nástroja je zostavovanie projektu. Nástroj zaisťuje vykonanie potrebných krokov pre zostavenie projektu, ako je kompilácia projektu, vytvorenie výsledného balíčku, automatická generácia dokumentácie alebo spustenie automatických testov.

6.2 Klientská aplikácia FITLayout4Web

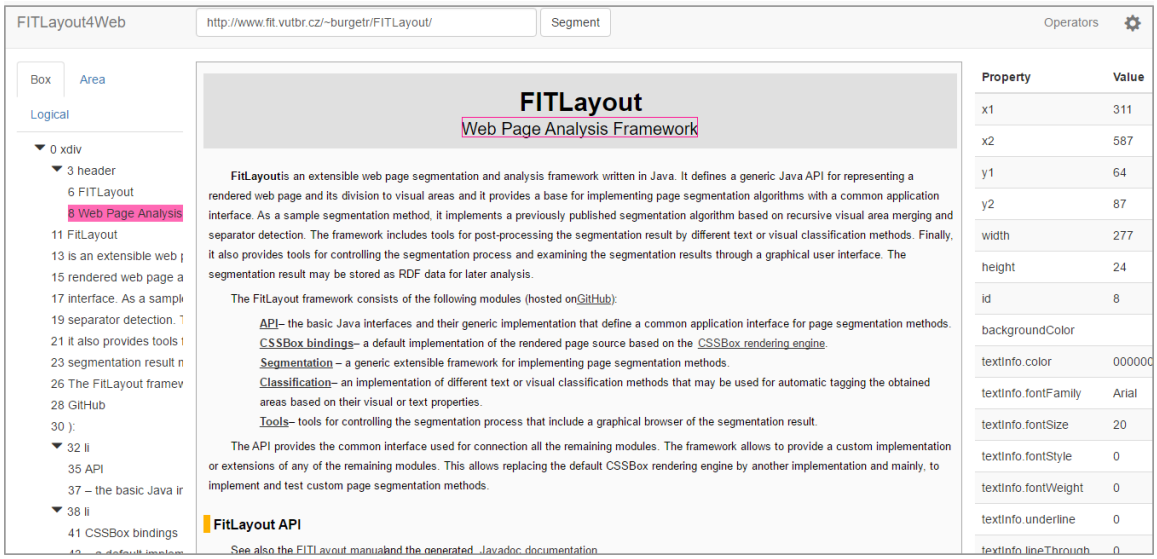
Bola vytvorená klientska aplikácia s webovým rozhraním typu klient-server, ktorá ponúka obdobné funkcie ako grafické nástroje dostupné vo FITLayout. Pre tvorbu klientskej časti bol použitý JavaScriptový framework AngularJS. Serverová časť bola implementovaná na platforme Java a bola použitá špecifikácia JAX-RS.

Klientska aplikácia FITLayout4Web umožňuje užívateľovi zadať URL adresu dokumentu, ktorý má byť analyzovaný. V hlavnej časti aplikácie je potom zobrazený tento požadovaný dokument, pričom tomuto dokumentu je možné zadať požadovanú šírku a výšku. Je možné vybrať vykresľovač, segmentátor, operátory a logický analyzátor, ktoré budú použité pri

[2] https://babeljs.io/
[3] https://www.npmjs.com/
[4] http://gulpjs.com/
[5] http://sass-lang.com/
[6] https://maven.apache.org/

analýze dokumentu. Aplikácia je schopná zobraziť strom boxov, strom vizuálnych oblastí alebo strom logických oblastí.

Štruktúru príslušného stromu je možné prechádzať a k jednotlivým uzlom sú vypísané vizuálne vlastnosti oblastí a ich hodnoty. Po kliknutí na uzol v stromovej štruktúre je zvýraznená príslušná oblasť v zobrazenom dokumente. Zároveň sú vypísané vizuálne vlastnosti z ich hodnoty tejto oblasti. Túto funkciu je možné vykonávať aj opačne, teda pri kliknutí na nejakú oblasť v zobrazenom dokumente v hlavnej časti okna je zvýraznený príslušný uzol v stromovej štruktúre. Pre zvýraznenie niekoľkých uzlov alebo oblastí súčasne je nutné držať tlačítko Ctrl.



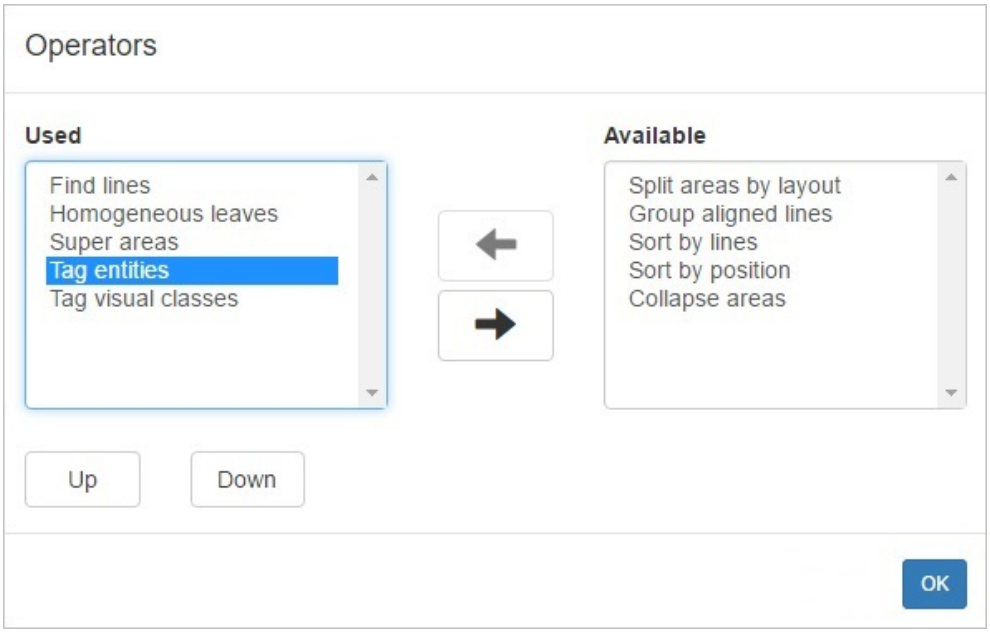
Obr. 6.1: Klientska aplikácia FITLayout4Web

Na obrázku 6.1 je možné vidieť snímku výslednej podoby implementovanej aplikácie. Aplikácia ponúka tieto funkcie:

1. V hornej časti okna je možné zadať URL adresu dokumentu, ktorý má byť analyzovaný.

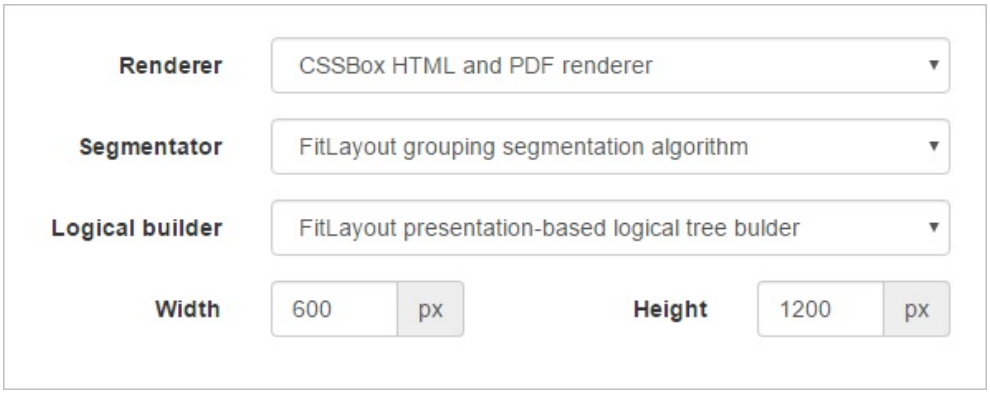
2. Kliknutím na tlačítko Segment dôjde k odoslaniu URL adresy, zvoleného vykresľovača, šírky a výšky dokumentu serveru. Server tento dokument načíta a spracuje. Následne je vytvorený strom boxov, ktorý je odoslaný naspäť klientovi. Klient potom tento strom boxov zobrazí ako stromovú štruktúru v ľavej časti okna a zároveň zobrazí analyzovaný dokument v strednej časti okna. Následne je serveru odoslaný zvolený segmentátor, zvolené operátory a strom boxov. Server s využitím zvoleného segmentátora vytvorí základný strom oblastí, na ktorý potom aplikuje zvolené operátory a vytvorený strom vizuálnych oblastí vráti klientovi. Klient tento strom zobrazí ako stromovú štruktúru v ľavej časti okna. Nakoniec je serveru odoslaný vybraný logický analyzátor a strom vizuálnych oblastí. Server na základe stromu a zvoleného analyzátora strom logických oblastí vytvorí a vráti ho klientovi. Aj v tomto prípade je zobrazená štruktúra stromu.

3. Pomocou tlačítka Operators sa otvorí dialógové okno, viď obrázok 6.2, kde je možné vybrať požadované operátory, ktoré budú použité pri segmentácii dokumentu, ich poradie a prípadné parametre.



Obr. 6.2: Klientska aplikácia FITLayout4Web - operátory

4. Pomocou ikony ozubeného kolieska si môže užívateľ vybrať zo zoznamu možných vykresľovačov, segmentátorov a logických analyzátorov viď obrázok 6.3. Tieto nastavenia potom budú použité pri analyzovaní dokumentu. Pre jednoduchosť použitia tento výber nie je nutný, pretože aplikácia ponúka prednastavené východiskové hodnoty. Zároveň je možné zadať požadované rozmery analyzovanej stránky.



Obr. 6.3: Klientska aplikácia FITLayout4Web - nastavenia

5. Záložky v ľavej časti obrazovky slúžia k výberu zobrazenia stromovej štruktúry stromu boxov, stromu vizuálnych oblastí alebo stromu logických oblastí.

6. V ľavej časti je zobrazená štruktúra požadovaného stromu. Pri kliknutí na ľubovoľný uzol bude zároveň zvýraznená odpovedajúca oblasť vo vykreslenom dokumente v strednej časti obrazovky. V pravej časti obrazovky budú vypísané vizuálne vlastnosti tohto uzlu a ich hodnoty.

7. V hlavnej časti obrazovky sa nachádza dokument vykreslený zvoleným vykresľovačom.

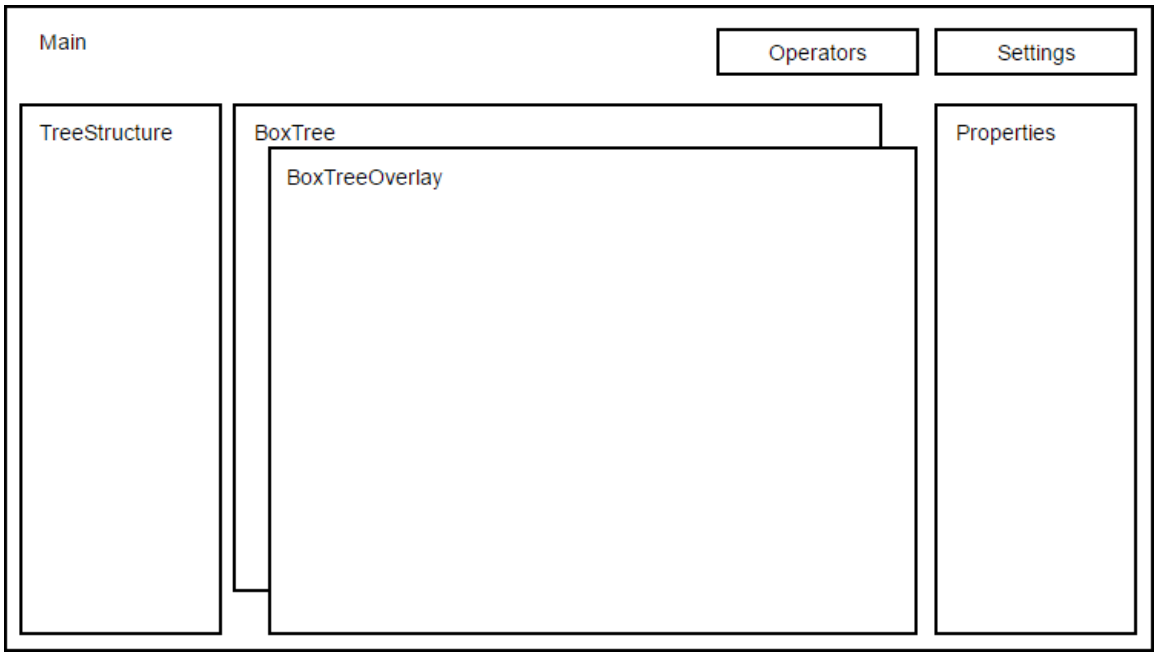
V dokumente je možné označovať ľubovoľné oblasti, ktoré budú zvýraznené aj v stromovej štruktúre a v pravej časti okna budú vypísané vizuálne vlastnosti tejto oblasti a ich hodnoty.

8. V tabuľke v pracej časti obrazovky sú zobrazené vlastnosti aktuálne vyznačenej oblasti alebo uzlu ako napr. veľkosť písma, farba a ich hodnoty.

6.3 Klient

Hlavnú súčasť výsledného produktu tvorí klientska časť webovej aplikácie, ktorá beží vo webovom prehliadači. Bol zvolený JavaScriptový framework Angular a grafická stránka aplikácie bola vytvorená použitím CSS frameworku Bootstrap. Angular umožňuje vývoj v jazyku JavaScript, TypeScript alebo Dart. Zo zadania diplomovej práce vyplynulo použitie jazyka JavaScript. Bola použitá najnovšia verzia jazyka ES2015. Celá klientska časť je tvorená komponentmi, viď obrázok 6.4, ktoré tvoria stromovú štruktúru:

- `MainComponent` - ústredný komponent,
- `TreeStructureComponent` - zobrazenie štruktúry stromu,
- `BoxTreeComponent` - vizualizácia stromu,
- `BoxTreeOverlayComponent` - ohraničenie vybraných oblastí,
- `PropertiesComponent` - zobrazenie vizuálnych vlastností vybranej oblasti a ich hodnôt,
- `OperatorsComponent` - výber operátorov, ktoré budú použité,
- `SettingsComponent` - možnosti nastavenia volieb pre spracovanie dokumentu.



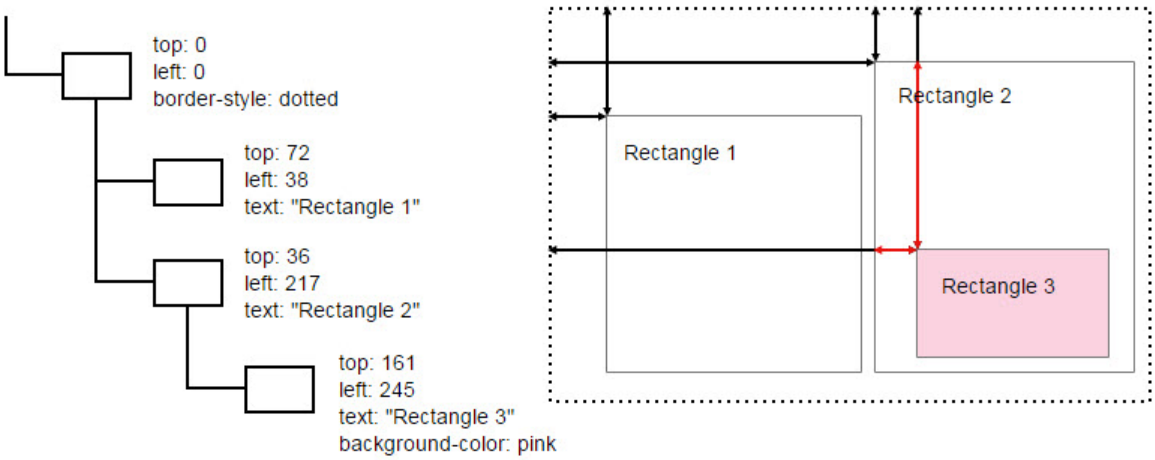
Obr. 6.4: Rozmiestnenie komponent

Koreňom tohto stromu je komponent `MainComponent`. Je to ústredný komponent, je rodičom všetkých ostatných komponentov a má priradenú šablónu, ktorá určuje celkové rozvrhnutie stránky. Zaisťuje tiež komunikáciu medzi svojimi potomkami a celkovo riadi aplikáciu. Súčasťou komponentu je vstupné pole, do ktorého je možné zadať URL adresu dokumentu, ktorý bude spracovaný. Komponent `MainComponent` je závislý na službe `TreeService`, ktorá je vložená do komponentu pomocou prostriedku vkladania závislostí (Dependency Injection). Za týmto účelom obsahuje trieda `Main` statickú vlastnosť `parameters`, ktorá určuje typ závislosti a na základe tejto vlastnosti Angular vloží inštanciu triedy `TreeService` do komponentu `MainComponent`. Pri zadaní URL adresy dokumentu a stlačení tlačítka Segment sa pomocou triedy `TreeService` získa strom boxov, strom vizuálnych oblastí a strom logických oblastí. Po prijatí odpovede komponent informuje svojich potomkov.

Komponenty `TreeStructureComponent` a `BoxTreeComponent` zobrazujú dáta, ktoré majú stromový charakter. Pri zobrazovaní dát sa komponenty rekurzívne zanárajú, pričom každému uzlu zdrojového stromu odpovedá jedna inštancia komponentu.

Komponent `TreeStructureComponent` zobrazuje štruktúru stromu boxov, stromu vizuálnych oblastí alebo stromu logických oblastí. Strom vizualizuje dáta vytvorené nástrojom FITLayout na serveri a odoslané vo formáte JSON klientovi. Výberom ľubovoľného uzlu kliknutím sa vyvolá udalosť, ktorá je propagovaná stromom smerom hore až ku koreňu. Koreňový uzol následne informuje svojho rodiča o udalosti (komponent `Main`). Na druhú stranu, strom umožňuje také udalosti aj prijímať, v takom prípade je udalosť šírená dole smerom k listom stromu. Ľubovoľný uzol stromu môže na udalosť reagovať - označenie alebo zrušenie označenia uzlu. Jednotlivé uzly je možné vyberať a pri stlačení tlačítka Ctrl je možné vybrať viac uzlov naraz. Po výbere uzlu v štruktúre strom vyznačí odpovedajúca oblasť v komponente `BoxTreeOverlayComponent` a zároveň sa zobrazia vizuálne vlastnosti a ich hodnoty danej oblasti v komponente `PropertiesComponent`. Pre lepšiu prehľadnosť je možné časti stromu rozbaľovať alebo zabaľovať.

Komponent `BoxTreeOverlayComponent` zachytáva udalosti kliknutia a na ich základe určí a zvýrazní konkrétny box, vizuálnu oblasť alebo logickú oblasť.



Obr. 6.5: Schematická ukážka komponenty Tree

Komponent `BoxTreeComponent` je najzložitejším komponentom, ktorý vizualizuje obsah stromu boxov. Vykresľovanie je založené na rekurzii, každý zanorený komponent predstavuje jeden uzol stromu. Schematická ukážka stromu a naznačenie súradnicového systému viď

obrázok 6.5. Vlastnosti uzlov sú využívané v šablóne komponentu a to tak, že zobrazený výsledok je vzhľadovo veľmi blízky zdrojovému dokumentu. Niektoré vlastnosti nie je možné použiť priamo, ale je nutné ich pred použitím transformovať. Ide napríklad o pozíciu uzlu. Uzly sú pozicované pomocou absolútnych pozícií v CSS. FITLayout používa súradnicový systém, v ktorom sú súradnice uzlu určované v rámci stránky vzhľadom ku koreňu. Zatiaľ čo v CSS sa súradnice uzlu určujú vzhľadom k najbližšiemu rodičovskému uzlu. Preto je potrebné pozície uzlov prepočítavať. Pri označení ľubovoľnej oblasti bude označený odpovedajúci uzol v komponente `TreeStructureComponent` a budú zobrazené vizuálne vlastnosti a ich hodnoty danej oblasti v komponente `PropertiesComponent`. Mapovanie všetkých použitých vlastností viď tabuľka 6.3.

Všetky vlastnosti vybraného uzlu sú zobrazované v tabuľke pomocou komponentu `PropertiesComponent`. O tom, ktorý uzol je aktuálne vybraný je komponent informovaný prostredníctvom udalostí.

Modálne okno `OperatorsComponent` je zobrazené po stlačení tlačítka Operators a umožní nastavenie operátorov a ich poradia, ktoré budú použité na základný strom vizuálnych oblastí pri segmentácii.

Pre nastavenia volieb pre spracovanie dokumentu slúži komponent `SettingsComponent`. V rámci tohto komponentu je možné vybrať zo zoznamu vykresľovačov, segmentátorov a logických analyzátorov. Tiež je možné nastaviť šírku a výšku viditeľného okna pri vykresľovaní.

Vlastnosť v CSS	**Vlastnosť v JSON**	**Popis**
`width`	`width`	šírka boxu
`height`	`height`	výška boxu
`top`	`top`	vzdialenosť boxu od horného okraja
`left`	`left`	vzdialenosť boxu od ľavého okraja
`background-color`	`background`	farba pozadia boxu
`font-size`	`fontSize`	použitý font textu
`color`	`color`	farba písma textu
`text-decoration`	`underline`	podčiarknutie textu
`text-decoration`	`lineThrough`	preškrtnutie textu
`font-weight`	`fontWeight`	tučnosť písma
`font-style`	`fontStyle`	kurzíva písma
`border-left-width`	`borders.left.width`	šírka ľavého okraja boxu
`border-left-color`	`border.left.color`	farba ľavého okraja boxu
`border-left-style`	`border.left.style`	štýl ľavého okraja boxu
`border-top-width`	`border.top.width`	šírka horného okraja boxu
`border-top-color`	`border.top.color`	farba horného okraja boxu
`border-top-style`	`border.top.style`	štýl horného okraja boxu
`border-right-width`	`border.right.width`	širka pravého okraja boxu
`border-right-color`	`border.right.color`	farba pravého okraja boxu
`border-right-style`	`border.right.style`	štýl pravého okraja boxu
`border-bottom-width`	`border.bottom.width`	šírka spodného okraja boxu
`border-bottom-color`	`border.bottom.color`	farba spodného okraja boxu
`border-bottom-style`	`border.bottom.style`	štýl spodného okraja boxu

Tabuľka 6.1: Použité vlastnosti

6.4 Server

Súčasťou implementácie klientskej aplikácie vo webovom prehliadači bolo vytvorenie serverovej časti. Bola použitá platforma Java EE vo verzii 7. Server v tomto prípade zabezpečuje dve hlavné úlohy, a to poskytnutie REST API pre prístup k službám nástroja FITLayout a distribúciu klientskeho obsahu.

Serverová časť poskytuje REST API, ktoré bolo implementované pomocou časti Java EE JAX-RS. Zdroje sú zoskupené do dvoch tried podľa charakteru. Jedna skupina je reprezentovaná triedou `OptionsResource` a obsahuje služby, ktorých úlohou je získavať informácie o možnostiach nastavenia parametrov nástroja, ako napr. dostupné vykresľovače alebo operátory. Všetky tieto zdroje podporujú iba HTTP metódu `GET`, takže všetky zdroje sú určené iba pre čítanie. Zdroje získavajú dáta od správcu služieb poskytnutého FITLayoutom. Tohto správcu predstavuje trieda `ServiceManager`. Druhá skupina je reprezentovaná triedou `TreeResource` a je tvorená službami, ktoré slúžia k získaniu stromu boxov, stromu vizuálnych oblastí a stromu logických oblastí. Tieto zdroje podporujú HTTP metódu `POST`. Zdroje spustia výpočet požadovaného stromu so zadanými parametrami a výsledok vrátia. Pretože triedy používané FITLayoutom sú príliš zložité na to, aby bolo možné jednoduché automatické mapovanie na JSON, bolo vhodné vytvoriť sadu jednoduchých tried, ktoré obsahujú rovnaké dáta a je ich možné automaticky mapovať. Tieto triedy majú jednoduchšiu štruktúru a mapovanie na JSON je jednoduché. Samotný proces mapovania zaisťuje aplikačný server.

Pre účely vývoja a testovania bol použitý aplikačný server WildFly[7].

Na nasledujúcej ukážke je možné vidieť ukážku dát stromu boxov vo formáte JSON, ktoré sú prenášané medzi serverom a klientom.

```
{
   "x1": 0,
   "y1": 0,
   "width": 1200,
   "height": 1200,
   "backgroundColor": "888888",
   "text": "text",
   "textInfo": {
      "color": "000000",
      "fontFamily": "Verdana",
      "fontSize": 13,
      ...
   },
   "visible": true,
   "type": "ELEMENT",
   "borders": [
      {"top": {"width": 0,"style": "SOLID", "color": "000000"} },
      {"left": {"width": 0,"style": "SOLID", "color": "000000"} },
      {"bottom": {"width": 0,"style": "SOLID", "color": "000000"} },
      {"right": {"width": 0,"style": "SOLID", "color": "000000"} }
   ],
   "childBoxes": [
      {"x1": 5,"x2": 10,"width": 25,"height": 25,"backgroundColor: "ffffff",
```

[7] http://wildfly.org/

```
            ...},
        {"x1": 100,"x2": 50,"width": 150,"height": 75,"backgroundColor:
            "fafafa", ...},
        {"x1": 505,"x2": 450,"width": 100,"height": 100,"backgroundColor:
            "ffffff", ...},
        ...
    ],
    ...
}
```

Každá oblasť je reprezentovaná podobnými dátami ako na ukážke, pričom potomkovia oblasti predstavujú obsah poľa `childBoxes`. Jednotlivé prvky v tomto poli majú rovnakú štruktúru ako rodičovská oblasť. Listové uzly, tj. uzly bez potomkov majú toto pole prázdne.

Kapitola 7

Testovanie a porovnanie funkčnosti

Na záver bolo vykonané testovanie implementovanej webovej aplikácie. Výsledný nástroj bol otestovaný na šiestich rôznych doménach a následne bol vizuálne porovnaný s výsledkami desktopového nástroja. Jednotlivé domény boli zobrazené v obidvoch nástrojoch a následne bola dôkladne porovnaná funkčnosť obidvoch nástrojov. Testovanie prinieslo niekoľko drobných rozdielov, ktoré sú popísané v podkapitole 7.1. V podkapitole 7.2 je možné vidieť ukážky obidvoch nástrojov a porovnanie funkčnosti.

7.1 Testovanie implementovaného nástroja

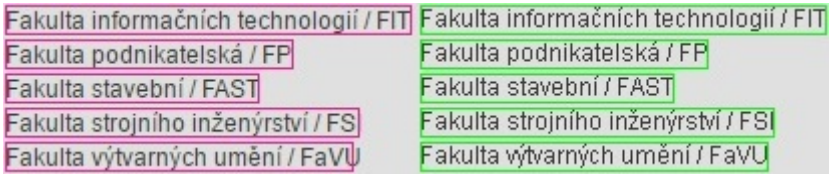
Implementovaná klientska aplikácia FITLayout4Web bola vizuálne otestovaná. Bola spustená vytvorená aplikácia spolu s desktopovou verziou nástroja. Postupne bolo vykreslených a detailne preskúmaných šesť rôznych webových stránok na obidvoch nástrojoch. Pri vizuálnom porovnávaní boli dôkladne sledované vykreslené dokumenty, porovnané všetky spoločné funkcie, ktoré nástroje ponúkajú. Testovanie prinieslo zistenie všetkých vizuálnych rozdielov alebo rozdielov vo funkčnosti. Vizuálna metóda testovania implementovaného nástroja sa ukázala byť v tomto prípade najviac efektívna. Ako vyplýva z podstaty implementovaného nástroja, automatické testovanie by v tomto prípade bolo veľmi obtiažne.

Klientska aplikácia bola otestovaná na týchto webových stránkach:

- http://cssbox.sf.net
- http://www.fit.vutbr.cz/
- http://www.vutbr.cz/
- https://en.wikipedia.org/wiki/Main_Page
- http://www.fit.vutbr.cz/~burgetr/FITLayout/
- http://www.fit.vutbr.cz/~burgetr/FITLayout/manual/

Podrobné porovnávanie jednotlivých stránok v implementovanej klientskej aplikácii a desktopovej verzii nástroja FITLayout neukázalo žiadne zásadné rozdiely. Pozícia aj rozmery jednotlivých prvkov na stránke boli rovnaké. Zhodné boli aj všetky ostatné vizuálne vlastnosti prvkov, ako farba pozadia prvku, použitý font písma, farba písma, veľkosť písma, modifikátory zobrazenia fontu (tučnosť, podčiarknutie, prešktrnutia a kurzíva písma) a nakoniec šírka, farba a štýl všetkých okrajov prvku.

Pri testovaní bol odhalený jeden rozdiel vykreslených stránok v implementovanej klientskej aplikácii oproti desktopovej verzii nástroja FITLayout. Rozdiel spočíval v odlišnosti fontov, čo môže mať za následok pretečenie obsahu oblasti, tj. textu z ohraničenia oblasti, ktorá je v obidvoch nástrojoch identická, viď obrázok 7.1.



Obr. 7.1: Rozdielnosť zobrazenia textu

Jednou z príčin môže byť skutočnosť, že klient a server nemusia mať k dispozícii tú istú sadu fontov. Server teda spočíta šírku boxu a určí, že v tomto boxe bude text určitého fontu. Môže ale nastať situácia, kedy klient tento konkrétny font nemá k dispozícii a zvolí nejaký alternatívny. Aj v prípade, že klient aj server majú k dispozícii tú istú sadu fontov, nemusia vyzerať vykreslené stránky zhodne. Ďalšou príčinou rozdielnosti zobrazenia textu môže byť iný spôsob vykresľovania textu. Programovací jazyk Java môže mať inú implementáciu ako webový prehliadač, napr. jedna implementácia uvažuje vyrovnávanie[1] (po anglicky kerning) alebo subpixelové vykresľovanie[2] (angl. subpixel rendering) a druhá nie.

Súčasťou testovania bolo aj porovnanie funkčnosti vo viacerých webových prehliadačoch, konkrétne boli použité Google Chrome, Microsoft Edge a Mozilla Firefox. Vyššie uvedená rozdielnosť textu sa vyskytla vo webových prehliadačoch Google Chrome a Microsoft Edge. Vo webovom prehliadači Mozilla Firefox bolo vykreslenie textu najviac podobné textu vykresleným v desktopovej verzii nástroja FITLayout. Príčinou je použitie rozdielnych implementácií pre renderovanie textu, napr. vo spôsobe vyhladzovania textu.

7.2 Porovnanie funkčnosti nástrojov

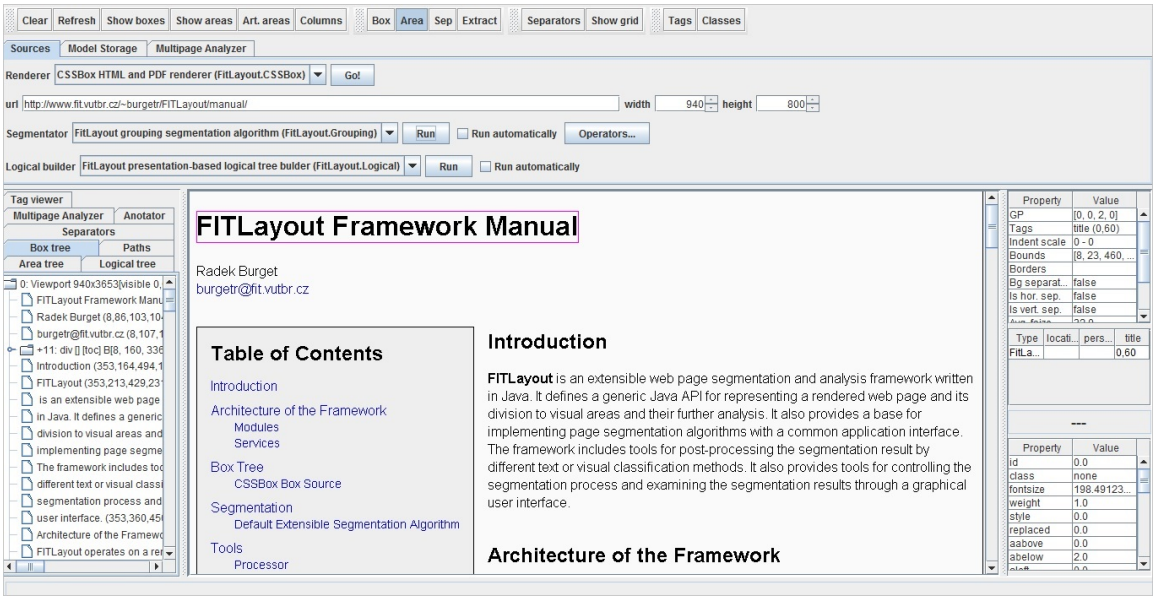
Z porovnania funkčnosti implementovanej klientskej aplikácie a desktopového nástroja vyplýva, že webová aplikácia nadobúda všetky dôležité funkcie ako východiskový desktopový nástroj. Pomocou webovej aplikácie je možné vykresľovanie rôznych webových stránok. Je možné vybrať zo zoznamu vykresľovačov, segmentátorov a logických analyzátorov. Rovnako je možné zadať rozmery vykresľovanej stránky. Je možné vybrať a zvoliť poradie operátorov, ktoré budú na základný vizuálny strom oblastí aplikovaný. Nad jednotlivými stránkami potom možno spustiť segmentáciu a následne zobraziť štruktúru stromu boxov, stromu vizuálnych oblastí alebo stromu logických oblastí. Výber oblasti na vykreslenej stránke sa prejaví zvýraznením odpovedajúceho uzlu v štruktúre daného stromu a naopak, výber uzlu zo štruktúry daného stromu sa prejaví ohraničením príslušnej oblasti vo vykreslenom dokumente. So zvýraznenou oblasťou sú zároveň vypísané vizuálne vlastnosti danej oblasti a ich hodnoty.

Na rozdiel od implementovanej webovej aplikácie desktopová verzia nástroja FITLayout umožňuje vytváranie rozširujúcich modulov, ktoré pridávajú ďalšie prvky do užívateľského prostredia, ako napr. práca s RDF úložiskom. Rozhranie pre tieto rozširujúce moduly je

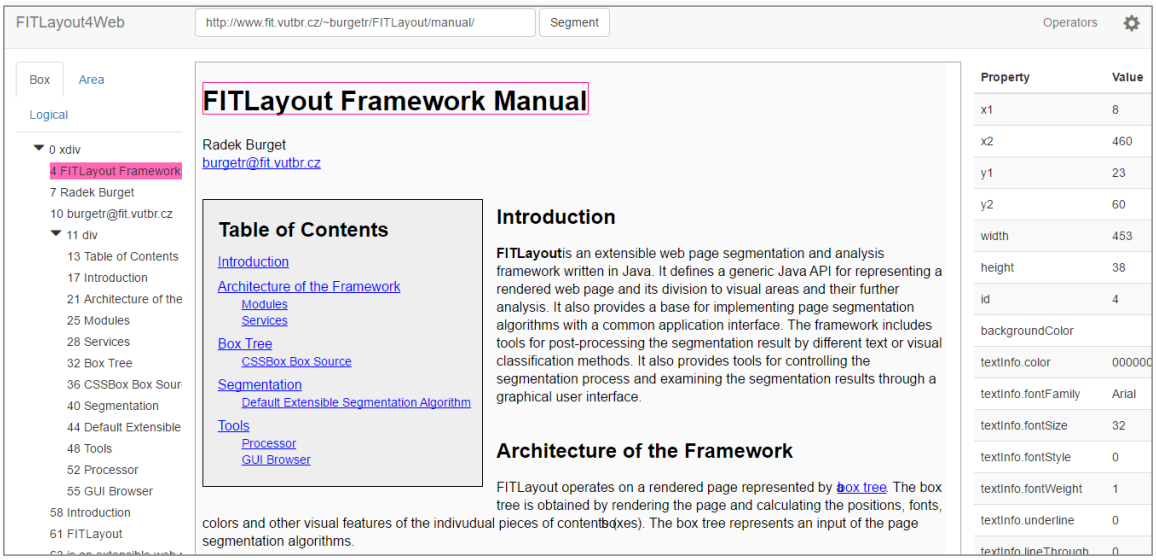
[1] Úprava medzier medzi určitými dvojicami písmen za účelom zlepšenia optického pôsobenia textu.
[2] Vyhladenie textu za účelom zlepšenia optického pôsobenia textu.

však úzko viazané na aplikačné rozhranie Java Swing a preto nie je možné podporovať tieto rozširujúce moduly vo webovej verzii nástroja.

Na nasledujúcich dvoch obrázkoch 7.2 a 7.3 je možné vidieť porovnanie grafického užívateľského rozhrania východiskovej desktopovej verzie nástroja FITLayout a implementovanej klientskej aplikácie FITLayout4Web.



Obr. 7.2: Desktopový nástroj



Obr. 7.3: Webová aplikácia

Kapitola 8

Záver

Cieľom tejto diplomovej práce bolo preskúmať existujúce rámce pre tvorbu klientskych aplikácií vo webovom prehliadači v jazyku JavaScript, zoznámiť sa s platformou Java so zameraním na tvorbu serverovej časti webovej aplikácie, zoznámiť sa s nástrojom FITLayout pre segmentáciu a analýzu dokumentov, navrhnúť architektúru s webovým rozhraním poskytujúcim obdobné funkcie. Následne implementovať navrhnutú webovú aplikáciu, otestovať implementovanú webovú aplikáciu a porovnať funkčnosť existujúcej desktopovej verzie nástroja FITLayout a vytvorenej webovej aplikácie. Tento cieľ bol splnený.

V kapitole 2 sa nachádza popis niektorých vybraných frameworkov pre tvorbu klientskych aplikácií vo webovom prehliadači v jazyku JavaScript. V kapitole 3 je popísaný architektonický štýl REST, platforma Java so zameraním na tvorbu serverovej časti webovej aplikácie a vybrané Java knižnice pre tvorbu REST služieb. V kapitole 4 je predstavený nástroj pre analýzu a segmentáciu webových stránok FITLayout. V kapitole 5 je navrhnutá architektúra webovej aplikácie a v kapitole 6 popísaná implementácia, konkrétne významné časti pri implementácii klientskej a serverovej časti. Nakoniec v kapitole 7 sa nachádza popis testovania vytvorenej webovej aplikácie FITLayout4Web a porovnanie funkčnosti východiskovej verzie nástroja s webovou aplikáciou.

Výsledkom je webová aplikácia typu klient-server poskytujúca dôležité funkcie nástroja FITLayout. Pre tvorbu klientskej časti webovej aplikácie FITLayout4Web bol použitý framework AngularJS. Serverová časť poskytuje REST rozhranie a bola vytvorená použitím platformy Java EE. Pomocou vybraného vykresľovača je možné zobraziť požadovanú webovú stránku, zobraziť strom boxov, strom vizuálnych oblastí a strom logických oblastí tejto webovej stránky. Štruktúru jednotlivých stromov je možné prechádzať a tak zvýrazňovať odpovedajúce uzly na vykreslenej stránke. Zároveň sú k vybraným uzlom zobrazované vizuálne hodnoty a ich vlastnosti.

Webovú aplikáciu by bolo možné rozšíriť vytvorením rozhrania umožňujúce tvorbu webových modulov, ktoré by pridávali ďalšie prvky do užívateľského rozhrania.

Literatúra

[1] Burget, R.: FitLayout - Web Page Analysis Framework [online]. 2014 [cit. 2015-29-12]. URL http://www.fit.vutbr.cz/~burgetr/FITLayout/

[2] Castro, E.; Hyslop, B.: *HTML5 a CSS3*. Computer Press, Albatros Media a.s., 2016, ISBN 9788025144657.

[3] Cravens, J.; Brady, T.: *Building Web Apps with Ember.js*. O'Reilly Media, 2014, ISBN 9781449370909.

[4] Fedosejev, A.: *React.js Essentials*. Packt Publishing, 2015, ISBN 9781782174622.

[5] Flanagan, D.: *JavaScript: The Definitive Guide*. Definitive Guides, O'Reilly Media, 2011, ISBN 9780596805524.

[6] Green, B.; Seshadri, S.: *AngularJS*. O'Reilly Media, 2013, ISBN 9781449355876.

[7] Jendrock, E.; Cervera-Navarro, R.; Evans, I.; aj.: Java Platform, Enterprise Edition The Java EE Tutorial, Release 7 [online]. 2014 [cit. 2016-01-02]. URL https://docs.oracle.com/javaee/7/JEETT.pdf

[8] Juneau, J.: *Java EE 7 Recipes*. Apress, 2013, ISBN 9781430244257.

[9] Konda, M.: *Just Spring*. O'Reilly Media, 2011, ISBN 9781449315467.

[10] Munro, J.: *Knockout.js: Building Dynamic Client-Side Web Applications*. O'Reilly Media, 2014, ISBN 9781491914328.

[11] Niemeyer, P.; Leuck, D.: *Learning Java*. O'Reilly Media, 2013, ISBN 9781449372491.

[12] Odell, D.: *Pro JavaScript Development: Coding, Capabilities, and Tooling*. Expert's voice in Web development, Apress, 2014, ISBN 9781430262695.

[13] Osmani, A.: *Developing Backbone.js Applications*. O'Reilly Media, 2013, ISBN 9781449328566.

[14] Spurlock, J.: *Bootstrap*. O'Reilly Media, 2013, ISBN 9781449344603.

Prílohy

Zoznam príloh

Príloha A

Obsah CD

Priložené CD obsahuje:

- technickú správu a jej zdrojové kódy,
- zdrojové kódy vytvorenej aplikácie.